# Modeling Group Communication in a Complex System for Achieving Group Goals

Paul Darbyshire

B.Sc (Hons), Grad Dip Computer Science, MEng

College of Business

Victoria University

Submitted in fulfillment of the requirements of the degree of

Doctor of Philosophy

(July, 2013)

Declaration of Candidate

"I, Paul Darbyshire, declare that the PhD thesis entitled *Modeling Group Communication in a Complex System for Achieving Group Goals* is no more than 100,000 words in length including quotes and exclusive of tables, figures, appendices, bibliography, references and footnotes. This thesis contains no material that has been submitted previously, in whole or in part, for the award of any other academic degree or diploma. Except where otherwise indicated, this thesis is my own work".

Signature 

Date 21/8/2014

# Dedication

In many respects the task of putting together a dedication for such a significant piece of work can be harder to begin than the work itself. There are so many people to thank for the supervision and help I have received along the way, that looking back, you realize how much you have relied on colleagues and family.

I would like to thank Professor Glenn Lowry (Retired) and Dr Bob McKay, formerly at the University of New South Wales for both getting me started on the PhD and for many interesting conversations. I would also like to thank Dr Justin Wang from Latrobe University and Dr Stephen Burgess from Victoria University for being a listening post over many years. In particular, I would like to thank my supervisors Professor Michael McGrath and Professor John Zeleznikow for all their help throughout the duration of the thesis. A special thanks John for picking up the pieces and the dedication and support over the last couple of years without which I am sure this would not have come to fruition. I am in debt to you.

Additionally, I would now like to thank those closest to me who have been with me on the many journeys I've undertaken to finish this work. My family, without whose support none of this would be possible. To my wife Kay who has stood with me and given support without question for the many long years this has taken, hopefully I can make up for this. To my son Adam for help and support over the years as I worked on this. It's your turn now. Finally to my beautiful daughter Mai-Ling, who always made me happy with her smiles. I love you all.

Paul

# ABSTRACT

This thesis investigates the effect of communication as a function of time on a multi-agent simulation based on a military distillation utilizing reinforcement learning for a group of agents. The original contribution to knowledge is a new model of cooperative learning developed as an enhanced Q-learning update function which also includes learning events communicated by other agents. Further contributions lie in the detailed analysis of simulation results establishing evidence of the cause-effect relationship between communication and improved performance. The improvement in performance is visualized by utilizing surface plot diagrams of the agents state-action matrix. These diagrams show the how group communications reinforce effective actions for the agents at an early stage in the simulation.

Classic reinforcement learning allows us to study the behavior of a single agent in an environment where the learning is governed by a reward function and is developed around a state-space conforming to a Markov assumption. In this environment the state signal observed by the agent is only influenced by the previous state signal. Reinforcement learning has been applied to multi-agent systems by a number of researchers with promising results. One class of simulation highly suited for use in multi-agent reinforcement learning is that of a military distillation. This allows us to model the cognitive ability of a team of agents, allowing them to adapt to the opposing team. Such a distillation represents a highly hostile environment and despite the inclusion of a learning ability, a team of agents utilizing a reinforcement-learning paradigm does not learn at a sufficient rate for agents to adapt to the hostile environment. Unless individual agents learn to survive they die, significantly affecting their team's chances of success.

We can increase the learning rate of the agents by allowing them to communicate between team members and thus increase the team's survival and success chances. Reinforcement learning has been utilized to investigate the distributed learning problem in many different multi-agent team-based scenarios. The communication between the agents then becomes a focal point of study when investigating group learning success. Invariably, using a distributed learning approach by allowing agents to exchange what they have learned the team does better in terms of achieving its goals. While the cause-effect relationship between communication and improved performance has been previously studied, it is difficult to find a detailed analysis of the effect the communication has as a function of time on the elements of the simulation. There is a gap in the literature on how this observed effect is realized within individual agents over time. This research addresses this gap in the literature.

# CONTENTS

.

**CHAPTER 1 INTRODUCTION.....1**

1.1 RESEARCH DESCRIPTION.....2

1.2 RESEARCH TOPIC AND CENTRAL RESEARCH QUESTION.....4

1.3 RESEARCH DESIGN.....6

1.4 CONTRIBUTION.....8

1.5 PUBLICATIONS.....9

1.6 THESIS LAYOUT.....11

**CHAPTER 2 LITERATURE REVIEW.....14**

2.1 HUMAN GROUP COOPERATION.....15

*2.1.1 Benefits of Groups Cooperation.....17*

2.2 AGENTS AND MULTI-AGENT SYSTEMS.....19

*2.2.1 Agency.....19*

*2.2.2 Multi-agent Systems.....22*

*2.2.3 Agent Situatedness.....24*

2.3 SIMULATION.....26

*2.3.1 Monte Carlo Simulations.....29*

*2.3.2 Continuous Simulations.....30*

*2.3.3 Discrete Event Simulation.....31*

2.4 AGENT BASED MODELING AND SIMULATION.....32

*2.4.1 Multi-Agent Based Simulations.....33*

*2.4.2 Simulation and Complexity*.....35
*2.4.3 Emergence*.....36
2.5 Combat Simulation.....38
2.6 Learning Agents in Simulations.....42
*2.6.1 Reinforcement learning*.....43
*2.6.2 Q-Learning*.....46
*2.6.3 Multi-Agent Reinforcement Learning*.....47
2.7 Cooperative Learning in Multi-agent Systems.....51
2.8 Neural Networks.....53
*2.8.1 Neural Networks and Reinforcement Learning*.....57

**Chapter 3 Methodology.....60**
3.1 Research Question Revisited.....60
3.2 The Research Process.....62
3.3 Research Methodology.....68
*3.3.1 Qualitative vs Quantitative*.....68
3.3.1.1 Action Research.....70
3.3.1.2 Experimental Research.....70
3.3.1.3 Case Study Methods.....71
3.3.1.4 Survey Methods.....73
*3.3.2 Simulation as a research Tool*.....74
3.4 Research Design.....80

**Chapter 4 Simulation Tool Design.....85**
4.1 Language and Versioning Details.....86
4.2 Architecture of the Simulation.....88
*4.2.1 Simulation Parameter Control*.....89
*4.2.2 Parameter Groups*.....91
4.2.2.1 World Parameter Group.....91
4.2.2.2 Simulation Parameter Group.....92
4.2.2.3 Agent Parameter Group.....93
4.3 The Simulation Engine.....95
*4.3.1 Sim Manager Architecture*.....97
*4.3.2 Action Manager Architecture*.....98
4.4 Representation of Time.....99
4.5 The Thinking Cycle.....102

4.6 Representation of State....106
4.7 Communication....109
4.8 Implementation of an Agents Weapon....114
4.9 Control Group Description....115
4.10 Simulation Output....118

**Chapter 5 Simulation Results....123**
5.1 Messaging and Learning Events....124
5.2 Model for Co-operative Learning....129
*5.2.1 Task Performance Gains....133*
5.3 Simulation Description....133
*5.3.1 Initial State Space....134*
*5.3.2 Reward Function....137*
*5.3.3 Initial Action List....138*
5.4 Experimental Design....139
5.5 Initial Benchmark Results....142
5.6 State Space Variables and Actions Refinement....145
*5.6.1 Action Refinements....148*
5.7 Experimental trials....150
*5.7.1 Benchmark Trial....151*
*5.7.2 Comms1 Trial....153*
*5.7.3 Comms2 Trial....154*
*5.7.4 Comms3 Trial....156*
*5.7.5 Initial Discussion....156*
5.8 Investigating Simulation Artifacts....162
*5.8.1 Agent Actuation Rate....163*
*5.8.2 Reward Rates....164*
5.9 Examining Agent Knowledge Landscapes....167
*5.9.1 Benchmark Trial....171*
*5.9.2 Comms3 Trial....179*
*5.9.3 Comms1 Trial....186*
*5.9.4 Comms 2 Trial....190*
5.10 Discussion....194

**CHAPTER 6 MODELING COMMUNICATION.....202**
6.1 COMMUNICATION IN A COMPLEX ADAPTIVE SYSTEM.....203
6.2 COMMUNICATIONS SYSTEMS VIEW.....204
6.3 MODELING COMMUNICATION WITH A NEURAL NETWORK.....206
6.4 DEFINING THE OBJECTIVE FUNCTION.....210

**CHAPTER 7 CONCLUSION.....213**
7.1 SUMMARY.....214
7.2 CRITICAL APPRAISAL.....218
7.3 CONTRIBUTION.....222
7.4 FURTHER RESEARCH.....223

**BIBLIOGRAPHY.....226**

**APPENDIX A SIMULATION CODE.....A1**
A.1 ARES APPLICATION MAIN CLASS.....A2
A.2 ARES APPLICATION MAIN GUI FORM CLASS.....A3
A.3 PARAMETER GROUP CHECKING CLASS.....A13
A.4 AGENT PARAMETER GROUP INPUT CODE.....A17
A.5 SIMULATION ENGINE CODE LISTING.....A26
A.6 SIM MANAGER CODE LISTING.....A29
A.7 SIM CLASS CODE LISTING.....A32
A.8 SIMMIND OBJECT CODE LISTING.....A34
A.9 ACTION MANAGER CODE LISTING.....A35
A.10 ACTION OBJECT CODE LISTING.....A37
A.11 BLUEMIND CODE LISTING.....A38
A.12 RLSTATEACTION OBJECT CODE LISTING.....A52
A.13 RLSTATE OBJECT CODE LISTING.....A53
A.14 RLUTIL OBJECT CODE LISTING.....A55

**APPENDIX B SIMULATION ELEMENTS.....B1**
B.1 STATE ACTION MAP.....B2

B.2 CRT OUTPUT FILE.....B4
B.3 CRE OUTPUT FILE.....B6
B.4 ACTVAL FILE OUTPUT.....B8
B.5 SCREEN FILE OUTPUT.....B11

**APPENDIX C PUBLISHED PAPERS.....C1**
C.1 ABSTRACTS.....C2

# Index of Figures

FIGURE 1.1: TRADITIONAL REINFORCEMENT LEARNING UPDATE CYCLE....8

FIGURE 2.1: EMERGING INTERDISCIPLINARY FIELDS....15
FIGURE 2.2: AGENT TYPOLOGY....20
FIGURE 2.3: 3D-SPACE MODEL OF AGENCY....21
FIGURE 2.4: CLASSIFICATION FOR MULTI-AGENT SYSTEM APPLICATIONS....23
FIGURE 2.5: SITUATED AGENTS FROM AN ORGANIZATIONAL PERSPECTIVE....25
FIGURE 2.6: MODELING VS. SIMULATION....27
FIGURE 2.7: MODEL REPRESENTATION OF A SYSTEM....27
FIGURE 2.8: SIMULATION ABSTRACTION LEVELS....28
FIGURE 2.9: REPRESENTATION OF DETERMINISTIC MODEL....29
FIGURE 2.10: REPRESENTATION OF MONTE CARLO SIMULATION MODEL....29
FIGURE 2.11: ELEMENTS MODELED IN MULTI-AGENT SYSTEM....35
FIGURE 2.12: TRADITIONAL REINFORCEMENT LEARNING UPDATE CYCLE....44
FIGURE 2.13: BIOLOGICAL NEURON ....54
FIGURE 2.14: GENERAL FORM OF AN ABSTRACT ARTIFICIAL NEURON ....55
FIGURE 2.15: FORM OF DIFFERENT ACTIVATION FUNCTIONS ....56
FIGURE 2.16: MCCULLOCH-PITTS EARLY MODEL OF A NEURON ....57

FIGURE 3.1: THEORETICAL CONSTRUCTS USED IN RESEARCH....64
FIGURE 3.2: THE RESEARCH CYCLE....65
FIGURE 3.3: SPIRAL DIAGRAM OF THE RESEARCH PROCESS....66
FIGURE 3.4: RELATIONSHIP BETWEEN ELEMENTS OF THE RESEARCH PROCESS....67
FIGURE 3.5: CYCLIC NATURE OF ACTION RESEARCH....70
FIGURE 3.6: FIGURE SHOWING THE CASE STUDY IN A FRAMEWORK METHODS USED TO REDUCE DATA COMPLEXITY....72
FIGURE 3.7: THE SURVEY PROCESS ....73
FIGURE 3.8: FRAMEWORK FOR SIMULATION METHODOLOGY....77
FIGURE 3.9: EXPANDED FRAMEWORK FOR SIMULATION METHODOLOGY ....78
FIGURE 3.10: GENERIC STAGES IN THE SIMULATION RESEARCH PROCESS....79
FIGURE 3.11: MAJOR ELEMENTS OF THE SIMULATION....81

FIGURE 4.1: OVERALL ARCHITECTURE OF THE SIMULATION....*88*
FIGURE 4.2: ARES GUI BASED MAIN MENU....89

FIGURE 4.3: PARAMETER GROUPS ACCESSIBLE FROM THE PARAMETER MENU ITEM.....91
FIGURE 4.4: STRUCTURE OF THE SIMULATION PARAMETER FILE.....92
FIGURE 4.5: SIMULATION PARAMETER INPUT FORM.....93
FIGURE 4.6: AGENT PARAMETER GROUP INPUT FORM.....94
FIGURE 4.7: ARCHITECTURE OF THE SIMULATION ENGINE AND COMPONENTS.....95
FIGURE 4.8: SCREEN CAPTURE OF A SIMULATION TRIAL IN PROGRESS.....96
FIGURE 4.9: REPRESENTATION OF THE SIM CLASSES WHICH MODEL AN AGENT IN THE SIMULATION .....98
FIGURE 4.10: AGENT ITERATION LOOP.....102
FIGURE 4.11: IMPLEMENTATION OF Q, STATE-ACTION MATRIX.....105
FIGURE 4.12: DEPICTION OF AGENTS SENSORY RANGES.....107
FIGURE 4.13: BUILDING THE AGENTS Q[ ] ARRAY FROM THE STATEACTIONMAP.....109
FIGURE 4.14: COMMUNICATIONS SENT AS ACTIONS IN THE SIMULATION.....113
FIGURE 4.15: CONTROL GROUP WEIGHTED BEHAVIORAL INPUT SCREEN.....116
FIGURE 4.16: CONTROL GROUP WEIGHTED INPUT SCREEN SECTION.....117

FIGURE 5.1: TRADITIONAL REINFORCEMENT LEARNING UPDATE CYCLE.....126
FIGURE 5.2: UPDATED REINFORCEMENT LEARNING INPUT SIGNAL.....129
FIGURE 5.3: AGENT TEAM INITIALIZATION.....140
FIGURE 5.4: PLOT OF DAMAGE OVER TIME FOR INITIAL BENCHMARK TRIAL.....143
FIGURE 5.5: INITIAL BENCHMARK TRIAL INDIVIDUAL SIMULATION RUN ANALYSIS.....144
FIGURE 5.6: DAMAGE VS TIME PLOT FOR UPDATED BENCHMARK TRIAL.....152
FIGURE 5.7: DAMAGE VS TIME PLOT FOR COMMS1 TRIAL.....154
FIGURE 5.8: DAMAGE VS TIME PLOT FOR COMMS2 TRIAL.....155
FIGURE 5.9: DAMAGE VS TIME PLOT FOR COMMS3 TRIAL.....156
FIGURE 5.10: COMPARATIVE BAR GRAPH SUMMARIZING Q-LEARNING AGENT TEAM WINS VS INSTINCTUAL AGENT TEAM WINS FOR ALL TRIALS.....157
FIGURE 5.11: Z TEST FOR 2 POPULATION PROPORTIONS.....158
FIGURE 5.12: COMPARATIVE GROUP LEARNING RATES OF THE Q-LEARNING TEAM (BLUE TEAM) FOR THE FOUR SIMULATION TRIALS.....160
FIGURE 5.13: ACTUATION RATE OF AGENTS IN THE Q-LEARNING TEAM FOR THE BENCHMARK COMPARED TO THE COMMS3 TRIAL.....163
FIGURE 5.14: COMPARATIVE PLOT OF THE RATE OF POSITIVE REWARDS RECEIVED FROM THE SIMULATION FOR THE Q-LEARNING BENCHMARK AND COMMS3 TRIALS.....165
FIGURE 5.15: COMPARATIVE PLOT OF THE RATE OF NEGATIVE REWARDS RECEIVED FROM THE SIMULATION FOR THE Q-LEARNING BENCHMARK AND COMMS3 TRIALS.....166

FIGURE 5.16: IMPLEMENTATION OF Q, STATE-ACTION MATRIX .....168
FIGURE 5.17: BENCHMARK TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - LEAST SUCCESSFUL AGENTS .....172
FIGURE 5.18: BENCHMARK TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - MOST SUCCESSFUL AGENTS .....174
FIGURE 5.19: COMMS3 TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - LEAST SUCCESSFUL AGENTS 180
FIGURE 5.20: COMMS3 TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - MOST SUCCESSFUL AGENTS. 182
FIGURE 5.21: COMMS1 TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - LEAST SUCCESSFUL AGENTS 187
FIGURE 5.22: COMMS1 TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - MOST SUCCESSFUL AGENTS. 189
FIGURE 5.23: COMMS2 TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - LEAST SUCCESSFUL AGENTS 191
FIGURE 5.24: COMMS2 TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - MOST SUCCESSFUL AGENTS. 193
FIGURE 5.25: BENCHMARK TRIAL CONTOUR DIAGRAM - MOST SUCCESSFUL AGENTS.....195
FIGURE 5.26: COMMS3 TRIAL CONTOUR DIAGRAM - MOST SUCCESSFUL AGENTS.....196
FIGURE 5.27: BENCHMARK TRIAL KNOWLEDGE LANDSCAPE DIAGRAM - MOST SUCCESSFUL AGENTS RESCALED.....197

FIGURE 6.1: COMMUNICATIONS SYSTEMS VIEW.....204
FIGURE 6.2: OVERVIEW OF NEURAL NETWORK MODEL WITH 5 NODES.....207
FIGURE 6.3: DETAIL OF NODES IN NEURAL NET MODEL.....208

# Index of Tables

TABLE 3.1: DISTINGUISHING CHARACTERISTICS OF QUANTITATIVE AND QUALITATIVE METHODOLOGIES ..... 69

TABLE 5.1: INITIAL STATE SPACE VARIABLES ..... 135
TABLE 5.2: REWARD FUNCTION FOR THE SIMULATION ..... 137
TABLE 5.3: INITIAL SET OF POSSIBLE ACTIONS ..... 138
TABLE 5.4: LIST OF PARAMETERS AND THEIR VALUES FOR SIMULATION TRIALS ..... 141
TABLE 5.5: MODIFIED LIST OF STATE SPACE VARIABLES ..... 146
TABLE 5.6: UPDATED SET OF POSSIBLE AGENT ACTIONS ..... 148
TABLE 5.7: BRIEF DESCRIPTION OF THE FOUR COMPARATIVE SIMULATION TRIALS TO MEASURE THE EFFECTS OF COMMUNICATION ON GROUP LEARNING AND SUCCESS ..... 150

TABLE 6.1: REWARD FUNCTION FOR THE SIMULATION ..... 211

# Index of Listings

Listing 2.1: Q-Learning algorithm....47

Listing 4.1: Simulation engine main loop....100
Listing 4.2: Sim Manager cognition loop....101
Listing 4.3: Reinforcement Learning agent think() method....103
Listing 4.4: ProcessReward() method listing....111
Listing 4.5: Format of the crt file....119
Listing 4.6: Format of the cre files....120
Listing 4.7: Format of the actval file....121
Listing 4.8: Format of the screen text file....122

Listing 5.1: Excerpt of State Action Map text file....149
Listing 5.2: State-action matrix of agent agent 27 in simulation run 271 – states 0 to 33....175
Listing 5.3: State-action matrix of agent agent 27 in simulation run 271 – states 34 to 67....176
Listing 5.4: State-action matrix of agent agent 19 in simulation run 271 – states 0 to 33....177
Listing 5.5: State-action matrix of agent agent 19 in simulation run 271 – states 34 to 67....178
Listing 5.6: State-action matrix of agent agent 26 in simulation run 203 – states 0 to 33....183
Listing 5.7: State-action matrix of agent agent 26 in simulation run 203 – states 34 to 67....184
Listing 5.8: State-action matrix of agent agent 15 in simulation run 203 – states 0 to 33....185
Listing 5.9: State-action matrix of agent agent 15 in simulation run 203 – states 34 to 76....186

# 1

# INTRODUCTION

*Every new beginning comes from some other beginning's end.*

– *Seneca (Roman Philosopher, mid 1st century AD)*

As humans we intuitively understand the benefits of cooperation between individuals, or between large groups of individuals. Indeed cooperative behavior has played a pivotal role in human evolutionary history. In their paper, "The Origins of Human Cooperation" (Bowles and Gintis, 2003), Bowles and Gintis note that populations which are composed of groups characterized by high levels of interaction between their members tend to be successful in evolutionary terms. They also indicate that group characteristics and the behavior of individuals in cooperative groups can have a synergistic effect. In such cases, the group becomes more than just the sum of its parts. The synergy created by cooperation can often mean that the achievements of the group is beyond the capabilities of the individuals. Indeed, societies and groups can be compared to organisms as they are composed of many individuals working together to increase their survival chances (Boyd and Richerson, 2005).

While it had long been known by early researchers that cooperative behavior within groups facilitates the achievement of goals more efficiently, it wasn't until almost mid twentieth century that the first detailed theory on the subject was established (May and Doob, 1937). There was a renewed interest in the study of groups in the 1970's stimulated by empirical research, particularly in the area of the use of groups for

cooperative learning. Further research showed that group cooperation promotes higher achievement and productivity than individual efforts (Johnson et al., 1981). Within the learning process cooperation takes on many forms, but often manifests itself as a form of communication. Individuals learn skills, techniques and sometimes environmental conditions by communicating with others. Indeed "communication is a cooperation problem", (Sterelny, 2008). The quintessential links between evolving cooperation and communication can be found in the modern structured learning environment we create for children. Although there has been some research effort to measure empirically the importance of communication and learning time on individual skills, the results have not been persuasive due to limited measurement techniques (Sterelny, 2008). However new technologies and advances in computer simulation techniques have evolved over the past decade which allow us to investigate these important issues using computer modeling.

Humans have been using simulation for a long time. Early cave paintings in the caves of Lascuax in France demonstrate the unique human ability to abstract and imitate reality. However, most simulation has its roots in military simulation, with early military simulation as simple as gladiators training with swords against wooden dummies. With the advent of computer technology, we are able to take a far more abstract approach to simulations, and we are not constrained by time. Many research methods must make assumptions regarding the nature of cause and effect within the system they are studying. However, with simulation techniques, researchers can assume the complexity of the system and study this relationship in detail by moving forward in time in a controlled manner (Dooley, 2002). Additionally simulations tend to be quantitative tools allowing us to collect empirical data on elements of the simulation, as the simulation progresses through time.

## 1.1 Research Description

In the last decade, agent-based simulations have been increasing in popularity as a research tool for studying a variety of problems from social and economic issues (Nakano, 2007) to elements of complexity theory (Saoud and Mark, 2007). The application of learning techniques applied to multi-agent systems is a growing field

(Claus and Boutilier, 1998), and an exciting, emerging area of research is Multi-agent Reinforcement Learning. Multi-agent reinforcement learning is concerned with how an agent can learn to act optimally in an unknown environment through trial and error interaction, and in the presence of other adaptable agents (Khoussainov, 2004). A multi-agent system, while generally Markov-like in nature is essentially a non-Markov decision process, as the interaction between the agents themselves and concurrent learning affect the state transition probabilities (Arai et al., 2000). Convergence is one of the measurements of performance often used in many applications utilizing traditional reinforcement learning techniques (Hirashima, 2008). While learning is not guaranteed to converge in a non-markovian environment, the main problem stems from the non-stationary nature of the task due to hidden variables caused by other agents affecting the environment. However, reinforcement learning, and in particular temporal difference (TD) algorithms such as Q-learning, can perform well in multi-agent systems (Preux et al., 2004).

In the coordination of team-based approaches to problem solving, a number of research publications have reported results using simple communication to share reinforcement rewards (Kelly and Keating, 1998, Mataric, 1998, Nunes and Oliveira, 2003, Sen et al., 1994, Tan, 1993). In most cases, communication of rewards was used to implement a low level cooperation among the agents which resulted in accelerated learning rates and improved group performance. The broad aspects of the 'cause-effect' relationship between agent communication (cooperation) and performance has been extensively studied. However, there still exists a gap in the detailed analysis of the effect that communication has as a function of time on the elements of the simulation. This information can be important for a new class of conceptual simulations utilizing intelligent agents. These conceptual simulations are often used for studying adaptive human behavior in complex systems, such as social, economic and military systems. For example, traditional simulation models of combat environments have sought linear solutions and are deterministic. But military conflicts (particularly land combat) possess almost all the main features of complex adaptive systems (Ilachinski, 1996), and if an opposing side is capable of adapting to the combat efforts of the other, then the resulting combat is unlikely to be linear. This is also true of all complex systems involving many interacting components. The interaction between agents allows for the adaption of those

agents to the changing environment.

A major complaint often leveled at such simulations is that they don't explain what happens at the micro level (van der Hoog, 2004). When we are modeling a system of living individuals, human or otherwise, we are effectively trying to model a complex system. A complex system is a system composted of many interacting components, where the overall system activity is non-linear. Thus traditional deterministic simulations were not wholly effective in modeling these complex systems where a butterfly effect, due to the interaction can often produce quite different outcomes. Multi-Agent based simulations allow us to take a non-deterministic approach to modeling complex systems. Using a multi-agent based approach, the individuals become the basic building blocks of the simulation (Mazher, 2001). We can then specify the interaction between the individuals and observe the effects as the simulation unfolds. The specification of the interaction between the components (or individuals) effectively describe, or model the behavior of the individuals in the system being simulated.

Multi-Agent Systems (MAS) are at the heart of modern conceptual simulations, and provide us with a mechanism to simulate the non-linear local interaction between the individuals. By studying the details of precisely how the performance gain is realized between cooperating agents at the micro level, we can relate the results back to the underlying domain theories. This could allow us to enhance any existing domain rules of practice to achieve better real-world outcomes.

## 1.2 Research Topic and Central Research Question

The purpose of this research is to study agent communication in a complex system, where the agents are cooperating to achieve a common goal. A complex system is utilized as the agent environment since this provides a more realistic foundation for simulations designed to study realistic scenarios at the conceptual level. We know from previous research that cooperating agents generally do better in achieving group outcomes. However, what is lacking in the literature is a detailed analysis of the effect the communication between cooperating agents has as a function of time on the elements of the simulation. Given a lack of detailed analysis in the literature on this cause-effect

relationship, the central question then becomes, exactly where and how the resultant effect is realized within the simulation? Once this is analyzed, further interesting questions arise related to the central question. If we study the effects of communication over time can we see the relationship between performance and communication as a linear relationship or otherwise? Are the effects cumulative or do they degrade over time as the simulation progresses and the landscape changes?

More formally, the central question can be framed as: if we denote the set of agents in a Multi-Agent System (MAS) by *X*, then a MAS with *n* agents is given by

$$X = \{x_1, x_2, \ldots x_n\}$$

Assuming communication is a function of discrete time, we denote a message between two agents $x_i$ and $x_j$ at time *t* as

$$m_{i,j}(t) : x_i \rightarrow x_j$$

then the total communication for *X* at time *t* is then given by

$$M(t) = \{m_{i,j}(t)\}$$

and the messages received by $x_j$ at time *t* is denoted by

$$M_j(t) = \{m_{i,j}(t)\}_{i=1\ldots n, i \neq j} \subseteq M(t)$$

Now we know from previous research that allowing agents to communicate generally increases the groups chances of success, thus the communication *M(t)* at time *t* must have a discernible effect on the group at time *t+1*, say $\Delta_{t+1} X$ . As the simulation progresses in time, we can denote the change in the group from time *t* to time *t+1* as

$$\Delta_{t+1} X : \; X_t \overset{M(t)}{\rightarrow} X_{t+1}$$

To analyze the effects of agent communication as time progresses, in particular, the effects on the group *X*, we consider the progression of changes $\{\Delta_t X\}_{t=1\ldots T}$ where the end of the simulation time is denoted by *T*. These changes will be realized in the individual agents, thus $\forall x_i \in X$, $\Delta_{t+1} X = \{\delta_{t+1} x_i\}$ where $\delta_{t+1} x_i$ are the resultant changes in $x_i$ at time *t+1* caused by $M_i(t)$. Thus the central research question then becomes a complete description (or analysis) of

$$\{\Delta_t X\}_{t=1\ldots T}$$

Additionally, we need to be able to visualize each of the $\Delta_t X$ descriptions in order to better understand the effects of communication. So a method to visualize these descriptions will also be developed. Once an analysis of the effect of communication is complete, a method to model the communication is then developed to provide a foundation for general understanding and formal reasoning about the dynamics of group communication. This has not been effectively achieved to date at this level for Multi-Agent Systems. A thorough understanding of the cause-effect relationship between communication and performance will enable us to design agent-based communication systems which maximize the learning potential.

## 1.3 Research Design

This research is designed to investigate the cause effect relationship between agent communication and group outcomes using a Multi-Agent based simulation approach. The architecture of the simulation is a non-supervised multi-agent system representing a military distillation with two opposing teams who combat for survival. The simulation world is a continuous world where agent movement is determined by a vector consisting of angle and distance (based on a speed variable). The simulation is modeling a specific problem and utilizes a finite state and action space. Time is discrete, and for each time step, each agent observes the world, determines its current state, and chooses an appropriate action to perform. All actions are performed at the end of each

time step, with the sequence randomized before execution to prevent bias.

The specific environment, the battlefield, was chosen as a battlefield contains all the elements of a complex system, many interacting components, with a non-deterministic outcome. The problem is generally well understood, and allows us to simulate a realistic scenario with possible feedback to the real, underlying domain. As mentioned, there are two groups of agents being simulated in the environment. One group represents the control group where the agents do not communicate, but have a common goal. The other group will represent the study group, also with a common goal, where the communication between the group members and its effects on the group outcome will be the focus of study.

Each of the individual agents will implement a reinforcement learning algorithm. Reinforcement learning is concerned with how an agent can learn to act optimally in an unknown environment through trial and error interaction with the environment and elements of the environment. There are many different reinforcement learning algorithms allowing many forms of learning. Most of these are episodic in nature with the simulation being performed many times and the agent learning taking place at the end of each episode. This type of Monte-Carlo approach is excellent for specific types of problems, but to simulate a more realistic scenario, a Temporal Difference reinforcement learning approach known as Q-learning (Sutton and Barto, 1998) was chosen. Q-learning is a well known reinforcement learning algorithm with the agent learning taking place as the simulation progresses.

Traditionally, reinforcement learning is for a single agent learning to perform a task in its environment. Utilizing Reinforcement learning in a multi-agent system requires adjustment to the algorithm. In particular, traditional reinforcement algorithms do not allow for the sharing of information or for communication with other agents. The traditional Reinforcement learning update algorithm defined in Sutton and Barto (Sutton and Barto, 1998) is based on the following cycle shown in Figure 1.1:

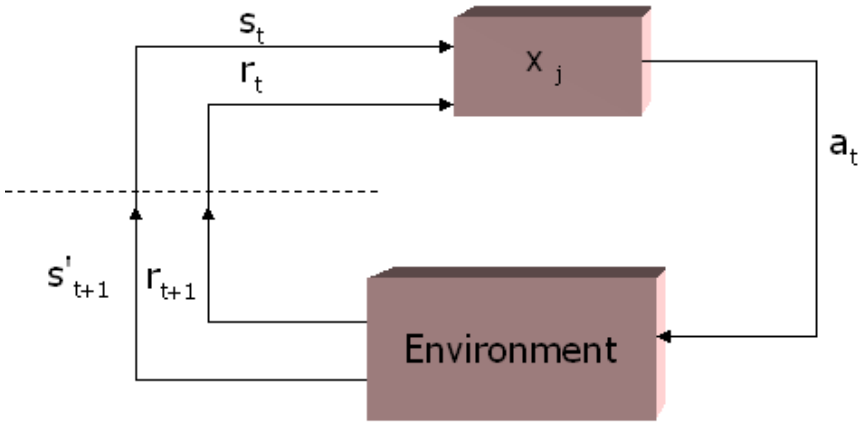


*Figure 1.1: Traditional reinforcement learning update cycle*

where $x_j$ is an agent, $s \in S$ (set of states), $a \in A_s$ (set of actions for state $S$), $r$ is the reward received and $s' \in S$ is the resulting state. Time is represented by $t$. However, with other agents communicating with each other at each time step, the traditional learning update cycle needs to be modified to accommodate the additional communication in the simulation.

A multi-agent based simulation package initially developed at the Australian Defence Force Academy was heavily modified to accommodate Reinforcement Learning agents developed for this research. It was also modified to allow for agent based communication, and the reinforcement learning updates to take place based on a modified Q-learning update equation developed for this research. Simulation runs using this multi-agent simulation tool were conducted with a control group and an experimental group to obtain the quantitative data used in this research. Additionally from this experimental data, an initial model for the communication which takes place within the complex system simulation, was developed utilizing a Neural Network based description.

## 1.4 Contribution

This research makes a number of contributions to the field of study.

A new model of cooperative learning is developed through the implementation of

an enhanced Q-learning update function which also includes learning events communicated by other agents. This is a modified version of the update function displayed in Sutton and Barto (Sutton and Barto, 1998). The original update function is for a single agent learning in an environment without the presence of other learning agents. The updated learning equation shown in the Methodological Approach above was specifically developed for this research. This updated function is a model that allows not only for the reinforcement learning agent to update its own learning events, but also allows further updates to the agent knowledge based on the the communication input signals from the other agents in the group. This model has been tested in the simulation with good results showing enhanced learning by the group.

Further contributions also lay in the detailed analysis of the simulation runs. This analysis pinpoints the effects of the communication on the agents, and visualizes these using knowledge landscape diagrams of the agents state-action matrix. These diagrams show how group communications reinforce effective actions for the agents at an early stage in the simulation. However, as the multi-agent system is non-markovian in nature, this becomes a moving target, and the surface diagrams evolve over time to reflect the changing nature of effective actions.

Finally, a Neural Network based model of the group communications is developed to effectively represent the communication in a more coherent form for further research. Such communication has always been difficult to represent at a micro level in a complex system, due to the non-linear nature of the systems. However, Neural Networks provide us with a good modeling tool to represent this information.

## 1.5 Publications

As a result of this research a number of Journal articles and conference papers have been published. All publications are blind peer reviewed papers and articles. The *Titles* and *Abstracts* of these articles and papers can be viewed in Appendix C. These are:

***Journal Articles and Chapters***

Darbyshire, P., Wang,D., "Effects of Communication in Cooperative Q-Learning", International Journal of Innovative Computing, Information and Control (IJICIC), Vol 6, No. 5, pp 2113-2126, May 2010, ISSN 1349-4198

Darbyshire, P., "Using XML to Help Isolate Software Systems and Agents from Change Due to Communications ", Journal of Business Systems, Governance and Ethics (JBSGE), Vol 1 No. 4, 2006, pp 59-68, ISSN 1833-4318

Darbyshire, P., Wang, D., "Learning to Survive: Increased Learning Rates by Communication in a Multi-agent System", Lecture Notes in Artificial Intelligence (LNAI 2903), Springer, ISBN 3-540-20646-9 pp 601-611, 2003

Darbyshire, P., "Effects of Communication on Group Learning Rates in a Multi-agent Environment", Advances in Complex Systems, Special Issue: Agent-Based Approaches in Complex Systems, Vol 6 No 3, September 2003, pp 405-426, ISSN 0219-5259

Darbyshire, P., "Modeling Agent Communication in a Complex System as a Neural Net", Lecture Notes in Engineering and Computer Science, Intentional Association of Engineers, Vol 2194 Issue 1, 2011, pp 1032-1037, ISSN 2078-0958

***Conference papers***

Darbyshire, P., "Using a Neural Net to Model Agent Communication in a Complex System", IEEE, Proceedings of the 2010 International Conference on Modeling, Simulation and Control, Cairo, Egypt, November 2010, pp 454-458, ISBN 978-1-4244-8823-0

Darbyshire, P., Wang, D., "What did they Learn?: the Effect on Learned Behavior of Increased Learning Rates in a Multi-agent System", 7th Asia-Pacific Conference on Complex Systems, Carins Australia, Dec 2004. pp 476-488 , ISBN 1-876674-96-2

Darbyshire, P., "Using XML for Simple Hierarchical Communication between Agents", IRMA' 2004, New Orleans, USA, May 2004, pp 686-689, ISBN 159140-279-4

Darbyshire, P., McKay, B., "Bringing Intelligence to the Battlefield", Modelling and Simulation (MS'02) 2002, Melbourne, Australia, 2002, ISBN 1-86272-617-5

Darbyshire, P., McKay, B, "Using Communication to Increase Learning in a Hostile Multi-Agent Environment", Complex Systems 2002, Tokyo, Japan, 2002

Darbyshire, P., Abbass, H., Barlow, M., McKay, B., "A Prototype Design for Studying Emergent Battlefield Behaviour through Multi-Agent Simulation", The 4th Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems, Hayama-machi, Japan, 2000

## 1.6 Thesis Layout

Chapter 2 of this thesis presents the literature review which provides the background of this research. The early interest of researchers into communication and in the development of a theory on human communication and how communication within groups can lead to more efficient goal achievement is explored. This phenomenon has also been observed in multi-agent simulations employing artificial intelligence techniques, but the cause-effect relationship has not been explored in detail. The history of simulation techniques is discussed which culminates in the agent-based techniques currently used by researchers to explore the non-linear local interaction between individual elements of a complex system simulation. We also present background literature on Complex Systems research and Reinforcement Learning, an Artificial Intelligence technique often used within multi-agent simulations. Additionally we also discuss Neural Networks and their relevance to Reinforcement Learning. Neural Networks will be used in a later chapter to Model the communication within the agent based simulations, however the work with Neural Networks does not form a core component of the research.

In Chapter 3 we discuss the methodological approach to the research undertaken for this thesis. The approach is essentially a quantitative methodology based on an experimental study to investigate a cause-effect phenomenon. The experimental study utilizes a discrete time simulation to model complex agent behavior. In this chapter, the use of simulations in research will be discussed and a formal framework for the research is developed.

Chapter 4 provides a detailed description of the project undertaken for this research. We discuss in particular the development of the simulation artifact used within the research to model the communication. We also provide a detailed explanation of the implementation of Reinforcement Learning agents used in the project, and their relationship to the theory outlined in Chapter 2. Given the many types of reinforcement learning algorithms, we also discuss the implementation methods used to facilitate the communication which takes place between the agents within the simulation. The different reinforcement learning algorithms will necessitate different techniques to share information.

In Chapter 5 we present the first of the results obtained from the research. In this chapter we discuss a model that was developed in this research to allow for communication to take place within a multi-agent system, based on reinforcement learning. The model was developed to facilitate communication in a non-episodic reinforcement learning simulation. This type of simulation best mimics reality, and provides for more realistic results which enable us to relate the simulation results back to the underlying domain. The model of communication allows the reinforcement learning agents to effect a multi-update on their action-value matrices instead of the single update per learning event, fundamental to the standard theory on reinforcement learning. This then allows the agents to learn from these learning events initially experienced by their peers. Using this model for communication we present the results of simulation runs and the effect that the communication has on group learning and the achievement of group goals. We also explore in detail these effects and pinpoint how the communication of learning events alters the reinforcement learning agents. To visualize these effects we use a variation of surface plot diagrams to visualize what the agents have learned over time.

In Chapter 6, after having detailed the effects of communication on the agents in the previous chapter, we develop a method for modeling the communication which takes place within a complex system, specifically the agent environment being utilized. We use a Neural Network to develop this model. A neural network provides a convenient mechanism to model the communication, with the agents represented an nodes in the neural network, the communication channels between the agents can be represented by weighted input vectors into the nodes. The weights on the input vectors then determine whether communication takes place at any specific discrete time within the simulation. By manipulating the weights in the neural network we can construct models of varying degrees of complexity.

In the final chapter, Chapter 7, we present and discuss the conclusions reached from the research undertaken in Chapters 4 and 5, We undertake a critical appraisal of the research presented in this thesis and relate the results back to the underlying domain of the simulation. Additionally we also outline some possible future directions of this research and present a plan or an immediate follow-up research project.

# 2

# LITERATURE REVIEW

*Only by not forgetting the past can we be the master of the future.*

– *Ba Jin*

Many areas of research are becoming increasingly interdisciplinary in nature which makes it difficult if not impossible to research effectively within the confines of a conventional scientific discipline (Arimoto and Ueta, 2009). This is based on the premise that increasingly, many of the research problems we face involve *Complex Systems*. A complex system is essentially one that is non-linear in nature and thus traditional linear solutions are not available. The composition and hybrid nature of many current research problems involves the integration of a number of traditional disciplines around the research question. This is necessary to tackle the problem from different perspectives which is often the best way forward (Conole et al., 2010) .

In their report "Emerging and Interdisciplinary Research Fields", Arimoto and Utes (Arimoto and Ueta, 2009) identify a number of emerging fields of research and show their links to existing traditional discipline fields. See Figure 2.1. The common element among these emerging fields is that fundamentally they are based on complex systems.

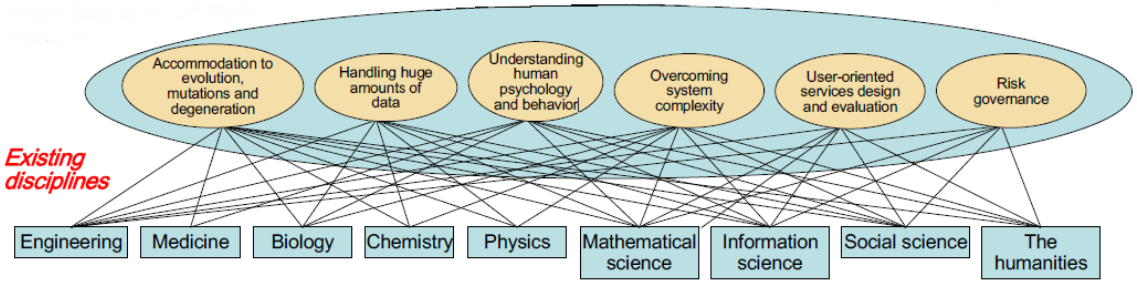


*Figure 2.1: Emerging interdisciplinary fields (adapted from Arimoto and Utea, 2009)*

This research is based on an emerging multi-disciplinary field, the Multi-agent System approach (MAS). The multi-agent system approach when combined with simulation techniques draws on many disciplines, including distributed artificial intelligence, artificial life, social simulation.

## 2.1 Human Group Cooperation

One of the more puzzling aspects of evolution is that organisms are essentially competitive, yet throughout nature we continually see widespread cooperation. In some animal societies we see extreme cooperation where a collective action arises as an emergent behavior from cooperation between the individuals (Rubenstein and Kealey, 2010). This is not an uncommon phenomenon and exists in not only flightless animal groups, but additionally in birds, and insect species such as ants and honeybees. However, in animal groups, complex cooperation only exists where all members of the group are closely related. While complex cooperation does exist in large societies of insects, these are eusocial societies where the individuals are closely related (Boyd and Richerson, 2005). In direct contrast to this almost all human societies are formed by the cooperation of large groups of unrelated individuals (ibid.).

Although instinctively humans are competitive in nature, we also understand the benefits of cooperation to achieve common goals. Indeed, cooperative behavior has played a pivotal role in human evolutionary history. In their paper, “The Origins of Human Cooperation” (Bowles and Gintis, 2003), Bowles and Gintis note that populations which are composed of groups characterized by high levels of interaction

between their members tend to be successful in evolutionary terms. They also indicate that group characteristics and the behavior of individuals in cooperative groups can have a synergistic effect. In such cases, the group becomes more than just the sum of its parts. The synergy created by cooperation can often mean that the achievements of the group is beyond the capabilities of the individuals. High-level comparisons are often made where societies and groups are compared to organisms as they are composed of many individuals working together to increase their survival chances (Boyd and Richerson, 2005).

Given the propensity of humans towards group cooperation, despite our inherent competitive nature, there must be advantages to the individual. There have been many attempts to explain this in terms of human and social evolution, but perhaps less subjective answers may be found in elements of *Game Theory* (Dresher 1981). Stephen Frank explains that the conditions required for an evolution towards group cooperation is that a change in the individual payoff with an increase in cooperation effort be greater than zero (Frank, 2009). This can be translated into the inequality:

$$rB_m - C_m > 0$$

in the above inequality $B_m$ is the *marginal benefit* term that measures the way in which a marginal increase in cooperation within the group affects marginal change in the individuals success. The quantity $r$ is applied as a weight to the marginal benefit term and measures the group behavior change rate relative to the individual behavior change. The term $C_m$ refers to the marginal cost of cooperation to the individual. Thus if the above inequality holds, then there is a positive payoff to the individual in group cooperation. This inequality is also a variation of the well known Hamilton's rule, which was used to detail the conditions for the evolution of altruistic behavior in organisms (Hamilton, 1963).

The above variation to Hamilton's rule suggests a marginal increase in payoff to the individual through group cooperation, but the inequality above has no constants and relies on terms such as marginal benefit, marginal cost and behavior change rate. These quantities become difficult to quantify, especially when applied to subjective human

behavior. Before Hamilton published his rule, earlier researchers established the first theories of cooperating groups.

### 2.1.1 Benefits of Groups Cooperation

It has long been known that cooperative behavior between groups of individuals facilitate the achievement of goals more efficiently, and sometimes allow groups of cooperating individuals to achieve goals beyond the capability of a single agent. Early 20th century researchers found that the quality of an individual's work increased when working in groups and that a group will 'think' more efficiently than the brightest individual of the group when alone (Gilles and Ashman, 2003). Not only did the quality of the individuals working in groups improve, but additionally the output of individuals within the group improved (Shaw, 1932). The early work by Shaw (*ibid*) was later duplicated in experiments where a waiting time model was developed to demonstrate the superior speed at which problems can be solved by group cooperation (Restle and Davis, 1962).

In 1937, May and Doob (May and Doob, 1937) established the first detailed theory on the subject of cooperative groups and studied the different behavior between groups and individuals. They found that groups of cooperating individuals were more successful in achieving goals than individuals working to achieve the same goal (May and Doob, 1937). There was a renewed interest in the study of groups in the 1970's stimulated by empirical research, particularly in the area of the use of groups for cooperative learning. Further research confirmed and extended the earlier work that group cooperation promotes higher achievement and productivity than individual efforts (Johnson et al., 1981). In fact, Johnson and Johnson (Johnson and Johnson, 1985) identified a number of variables that potentially affect the relationship between cooperation, productivity, and inter-group dynamics. These were clustered into three groups, Cognitive Processes (quality of learning), Social (mutual support) and Instructional (type of task). Much of this is difficult to quantify given the subjective nature of the variables, and there was not universal agreement between social scientists on the conclusions of such studies. But what was generally agreed upon was that groups had an effect on the individuals.

This "effect" of the group on the individual becomes an important aspect to study, given that groups of cooperating individuals generally fare better. Indeed, not only are groups that cooperate more efficient, but can also achieve goals that individuals cannot. The early work of Shaw (Shaw, 1932), described experiments where problems were given to individuals and small groups of individuals that cooperated. The problems were designed so that it was almost impossible for an individual to arrive at an early solution, but rather the problems were better solved with an interchange of ideas. The outcomes of these experiments showed that when a number of these experiments were conducted, the groups were far more likely to return a correct solution than the individuals. Examples of problems where groups of cooperating individuals are needed to achieve goals and solve problems can be found through history, such as exploration, warfare, scientific achievements and building civilizations. We can see smaller more mundane forms on a daily basis in government, business, large projects and almost any area where humans work together. The group has an effect on the individuals and enhances the learning process (Gilles and Ashman, 2003, Johnson and Johnson, 1985, Shaw, 1932, Dörnyei, 1997, Gokhale, 1995).

Within the learning process cooperation takes on many forms, but often manifests itself as as a form of communication. Individuals learn skills, techniques and sometimes environmental conditions by communicating with others. Indeed "communication is a cooperation problem" (Sterelny, 2008). The quintessential links between evolving cooperation and communication can be found in the modern structured learning environment we create for children. Although there has been some research effort to measure empirically the importance of communication and learning time on individual skills, the results have not been persuasive due to limited measurement techniques (Sterelny, 2008). However new technologies and advances in computer simulation techniques have evolved over the past decade which allow us to investigate these important issues using computer modeling and simulation in conjunction with Multi-Agent Systems.

## 2.2 Agents and Multi-Agent Systems

Over the last decade. the term *Agent* has been increasingly used by computer science researchers, engineers and software developers and has even crept into general user terminology. Yet when pressed, many would be unable to give a satisfactory explanation of just what an Agent really is. This can be readily forgiven, as even the experts cannot agree on a definition for an agent (Wooldridge and Jennings, 1995, Nwana, 1996). Agent software research was an emerging technology in the late 1990's, which promised to be many things to many people, however while the technology is no longer in its embryonic stage, it can still be considered an emerging discipline. Over the last decade, the programming languages and the maturity of the support technology used to build agents has developed to a level where agents have migrated from the laboratory to real world applications. The more recent notable applications for agents include e-Commerce, Web marketing and Internet search agents. However, apart from the more popular applications of the discipline, Agents and Multi-Agent systems are used extensively in military and social simulations.

In the following section the essence of *agency* is explored and a classification is provided. The notion of autonomous agents somewhat expands on the base idea of agency and is also discussed.

### 2.2.1 Agency

Agent technology emerged from the field of AI research, so often the term *Intelligent Agent* is used. However, agents need not be intelligent, in fact most people do not need 'smart agents' (Nwana, 1996). Other adjectives often used to describe agents are, interface, autonomous, mobile, Internet, information and reactive. The term *agent* can be thought of as an umbrella under which many software applications may fall, but is in danger of becoming a noise term due to over use (Wooldridge and Jennings, 1995).

What makes agents different from standard software is the characteristics that agents must possess in order to be termed agents. There are a number of classifications

schemes that can be used to type-cast existing agents, for example mobile or static, deliberative or reactive, but Nwana (Nwana, 1996) classifies agents according to primary attributes which agents should exhibit, see Figure 2.2. The three primary attributes are cooperation, learning and autonomy. These attributes are laid out as intersecting circles in Figure 2.2, and to be classified as an agent, software must exhibit characteristics from any of the intersecting areas as shown.

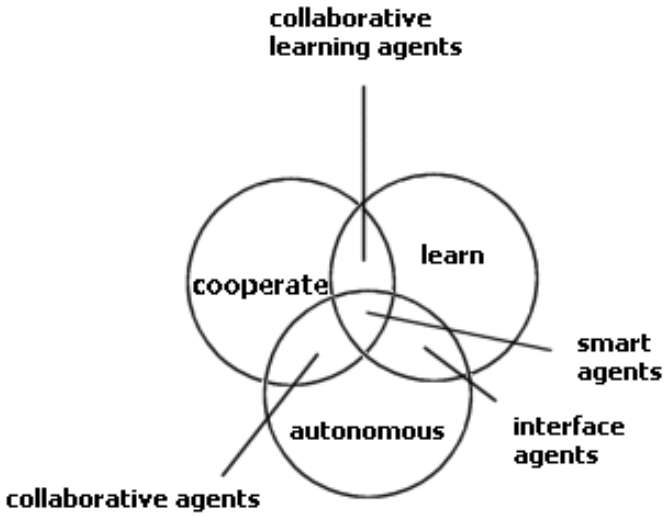


*Figure 2.2: Agent typology (Nwana, 1996)*

Nwana uses the diagram in Figure 2.2 to derive four types of agents, Collaborative, Interface, Collaborative Learning and Smart agents. However, Nwana recognizes that the categories in the diagram are not definitive and agents can also be classified by their roles, and so adds to the list Mobile, Information/Internet, Reactive and Hybrid agents.

Wooldridge and Jennings (Wooldridge and Jennings, 1995), take a more formal approach to the definition of agent, falling back to the more specific meanings from AI researchers. However, they note that as the AI community cannot agree on the question of "*what is Intelligence*?", a less formal definition may be needed to include many software applications being developed by researchers in related fields. To this end, Wooldridge and Jennings introduce the notions of weak and strong agency. Strong agency takes on the specific meaning from AI research, implying that agents must exhibit mentalistic notions such as knowledge, belief, intention and obligation, with

some researchers considering emotional characteristics as a requirement. If this definition of agent is strictly adhered to, many software applications claiming to use agent technology would be rejected as such.

In the weak notion of agency, the term agent can be applied to software that exhibits the following characteristics:

- Autonomy: agents operate without direct human intervention and have some control over their actions and internal state.
- Social Ability: agents interact with other agents and humans through some defined protocol.
- Reactivity: agents can perceive their environment and can respond to it in a timely fashion.
- Pro-activeness: agents do not just respond to the environment, but can take a proactive role and exhibit some goal-oriented behavior.

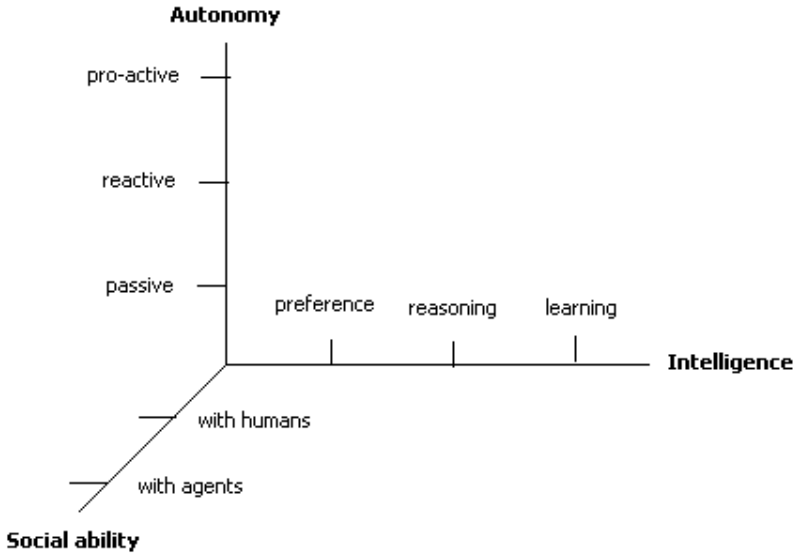


*Figure 2.3: 3D-Space model of agency*

The definitions of Nwana and Wooldridge above are not wholly incompatible. They do identify common characteristics which agents should exhibit, but most of the agent types identified by Nwana would come under Wooldridge and Jennings weak-agent classification. The characteristics of autonomy, cooperation, intelligence,

reactivity and pro-activity have also been identified by other researchers. Gilbert et.al. (Gilbert et al., 1995) provide a model where the degree of agency can be crudely measured by an agents position in a three-dimensional space relative to a 3-D axis. This model has been refined from Odell (Odell, 2010 pp58) with the refined version using the three dimensions of Intelligence, Autonomy and Social Ability defined from the list above. This model is shown in Figure 2.3.

In order to qualify as an agent in this model, software must exhibit at least the minimal characteristics in each dimension. That is, it must be able to communicate with the user, allow users to specify preferences, and be able to work out how to accomplish tasks assigned to it. While this model may not be ideal, it does provide for a more simplistic definition based on the common agent characteristics identified by a number of researchers.

### 2.2.2 Multi-agent Systems

As the technology matures and we apply agent based solutions to increasingly complex problems, we find an obvious need to develop solutions involving more than one agent (Sycara, 1998). One of the most powerful tools we have to reduce complexity is modularity. Building an agent based systems with more than one agent (Multi-agent System), allows us to introduce modularity into a solution. Further, as the problems become more complex we can address the modularity by increasing the number of agents in the system designed to address the solution. Thus, for many complex problems. a multi-agent system approach provides us with a natural technique which is immediately scalable to suit the complexity of the problem. Depending on the problem, the multi-agent system can he highly homogenous in nature, or heterogeneous if different agent types are needed for different aspects of the problem at hand.

The Multi-Agent System approach draws on many disciplines, in particular those of Distributed Artificial Intelligence, and Artificial Life (Ferber, 1999, Gilbert and Troitzsch, 2005). The driving force behind Distributed Artificial Intelligence is to create organizations of systems capable of solving problems by cooperative reasoning. Thus displaying a form of Artificial Intelligence generated from the synergy of the

organization. The intention of Artificial Life (ALife) is to understand systems that possess life (i.e. survive, adapt and reproduce), by modeling them and the environment they are situated in. These environments are sometimes hostile, such as a battlefield.

Multi-agent systems differ from single-agent systems in that several agents exist which model each others goals and actions (Stone and Veloso, 1997). From a single agent's perspective, a multi-agent systems differs from a single-agent system in that the environment's dynamics can be determined by other agents. However, Ferber (Ferber, 1999) gives a more precise definition, and defines a multi-agent system as one comprising the following elements:

- "*An environment, E, that is a space and generally has a volume.*
- *A set of objects, O, that are situated (by position) in E.*
- *An assembly of agents, A, which are specific objects ( $A \subseteq O$ ), representing the active entities of the system.*
- *An assembly of relations, R, which links objects and agents to each other.*
- *An assembly of operations, Op, making it possible for the agents of A to perceive, produce, consume, and manipulate the objects from O.*
- *Operators with the task of representing the applications of these operations, and the reaction of the world to this attempt at manipulation, which are the laws of the agent's universe.*"

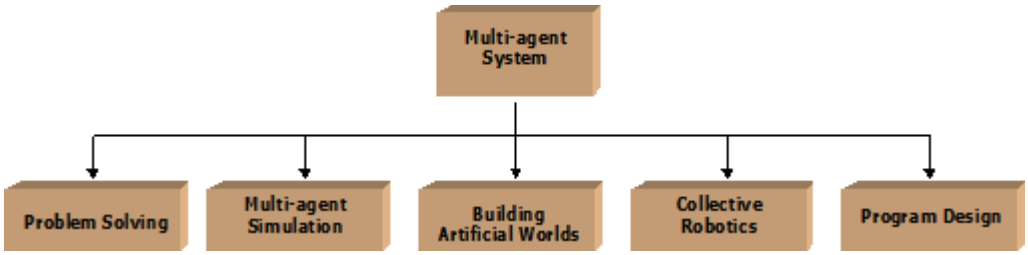


*Figure 2.4: Classification for multi-agent system applications*

There are many areas in which multi-agent systems may be applied, and as the discipline is evolving, new areas are being added. However, Ferber classifies these applications into five main categories, and these are shown in Figure 2.4. One of the

areas of intense interest by researchers is that of multi-agent simulation. We will discuss Multi-agent simulations in detail in section 2.4 , but they bring a new paradigm to the field of modeling and simulation, particularly in systems that involve simulation of living entities. By using a multi-agent system we have the possibility of directly representing individuals and their interaction in a simulation. In fact, central to the above definition of a multi-agent system are the concepts of an environment, and the interaction (relations) between agents. What characterizes a multi-agent system from other approaches is the dynamics of interaction between agents in the system (Ferber, 1999, Stone and Veloso, 1997, Gilbert and Troitzsch, 2005)

### 2.2.3 Agent Situatedness

The concept of agent situatedness has become important as agent based simulations (discussed in a later section) modeling agents in specific environments have become important tools for researchers (Kannengiesser and Gero 2005). Zambonelli Jennings and Wooldridge (Zambonelli et.al 2005) describe agent situatedness as being a key characteristic of a Multi-Agent System. Agents that exhibit the characteristic of "*situatedness perform their actions while situated in a particular environment. The environment may be a computational one or a physical one and an agent can sense and effect some portion of it*" (Zambonelli et.al 2005). This definition somewhat updates the Autonomy axis of the 3D-Space model of agency shown in Figure 2.3.

The concept of situatedness then invokes other important notions such as autonomy and implies design characteristics such as Memory and State. If an agent is situated in an environment, then it must be able to interact with that environment by choosing to perform actions that affect the environment and hence its own future interactions with the environment. The agent must be able to model the environment, either fully or partially, thus retain some memory of that environment. In order for the agents to choose meaningful actions when interacting with the environment, there must be a state space which allows the agents to perceive something about the environment in order for it to choose an appropriate action.

With the characteristic of *situatedness*, agents become coupled to the environment and all their actions and goals become oriented to the environment itself. For instance, choosing actions that affect the environment, cooperate with other agents in the environment, perceiving the environment, modeling the environment or moving about the environment in some way,

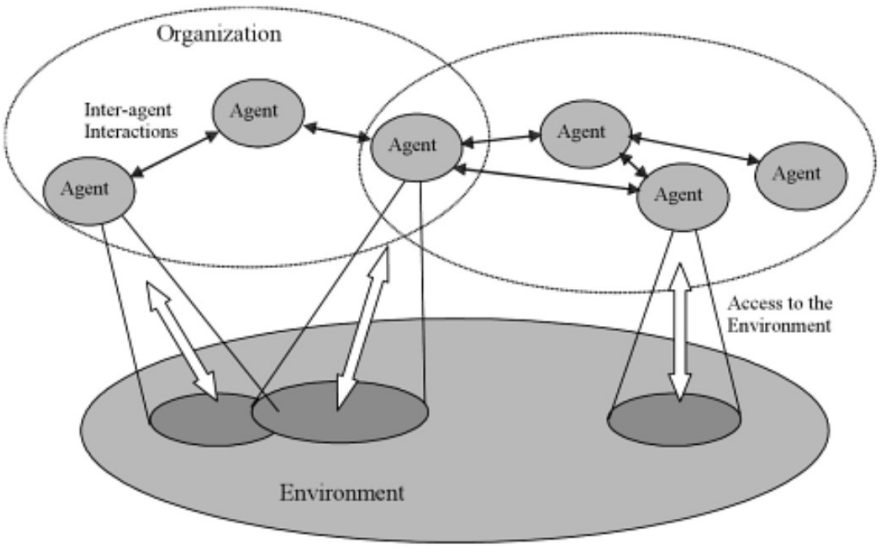


*Figure 2.5: Situated agents from an organizational perspective*

Figure 2.5 (Jennings 2001), show situated agents using an Organizational metaphor. Agents can be viewed as situated is a specific environment with other agents and belonging to an organization. It is possible for agents to belong to one or more organizations, and agents can interact with other agents and the environment itself. The organizational metaphor provides us with a powerful framework to utilize Multi-agent systems for studying complex real-world problems. In these important underlying domains, we are utilizing situated agents to model behavior in a specific environment.

## 2.3 Simulation

Humans have been using simulation for a long time. Early cave paintings in the caves of Lascuax in France demonstrate the unique human ability to abstract and imitate reality. However, most simulation has its roots in military simulation, with early military simulation as simple as gladiators training with swords against wooden dummies. With the advent of computer technology, we are able to take a far more abstract approach to simulations, and we are not constrained by time. The recent history of computer simulation began with John Von Neumann and Stanislaw Ulam working on the Manhattan Project during World War II (Balachandran et al., 2000). Despite this beginning, computer simulation was not regarded as a very useful tool in the 1950's as computer models were expensive to develop and results were too long in coming (Reitman, 1988). However during the 1960's, with developments and research at IBM Corporation, along with a number of conferences taking place, simulation began to be regarded as an important tool for understanding and prediction (Balachandran et al., 2000, Reitman, 1988).

With the increasing sophistication of computer technology we have seen a rise in the development of *computer models* and thus *computer simulation* for real-world understanding. In particular, computer simulations allow us to quantify the performance of a system we are studying using different values for the input parameters. This provides valuable information to assist in decision making processes (Perros, 2009). There are a number of benefits in using computer simulations: we can conduct "what-if" type scenarios by varying input parameters; many alternatives can be tested before choosing the best option in the case of some simulations; results can be produced and analyzed in less time than real-world observations; we can maintain better control of the experimental conditions (McHaney, 1991). Additionally simulations tend to be quantitative tools allowing us to collect empirical data on elements of the simulation, as the simulation progresses through time.

Often the terms *Simulation* and *Model* are used interchangeably, but there is a subtle difference. A Model is an abstract representation of a "*System*" in the real world. The System can be either natural or artificial, such as an ecosystem or a financial

system. A Simulation is the use of a Model to to formulate conclusions and provide some insight on the underlying real-world system, see Figure 2.6. The application of the simulation results in output which we can observe and analyze to reason about the underlying physical system.

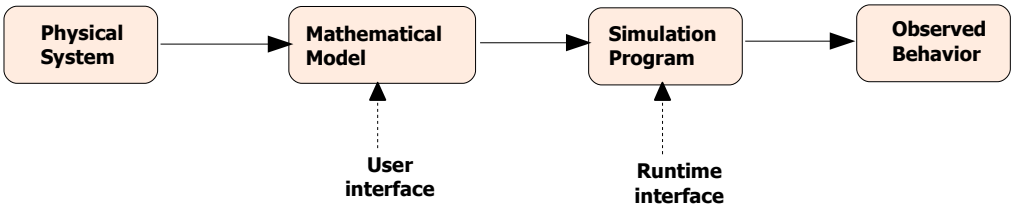


*Figure 2.6: Modeling vs. Simulation (adapted from Cellier and Kofman, 2006)*

The model is never a prefect representation, but rather an abstract representation of the system, where unimportant details are abstracted out. This is depicted in Figure 2.7, adapted from (McHaney, 1991).

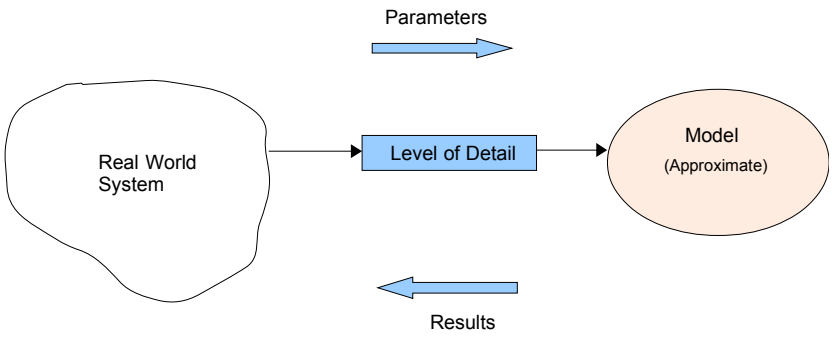


*Figure 2.7: Model representation of a System*

When we develop models of systems for simulation, the entire system is never simulated, rather, the simulation or model is a scaled down version of reality. This is necessary in order to reduce complexity and increase understanding. Thus, the model only includes as much detail of the real world system as necessary in order to study the aspects in which we are interested and produce results that correlates with the real world system. We can model the elements of the system we are including at varying levels of

detail in order to increase or decrease the abstraction level of the model. This is depicted in Figure 2.8 which was adapted from Perros (Perros, 2009). If the model we develop requires more detail, we can model the included elements from the real-life system in further detail. Modeling the elements of the system in less detail increases the abstraction level. Simulations at varying levels of abstraction are useful for gaining different levels of insights into the study of a system.

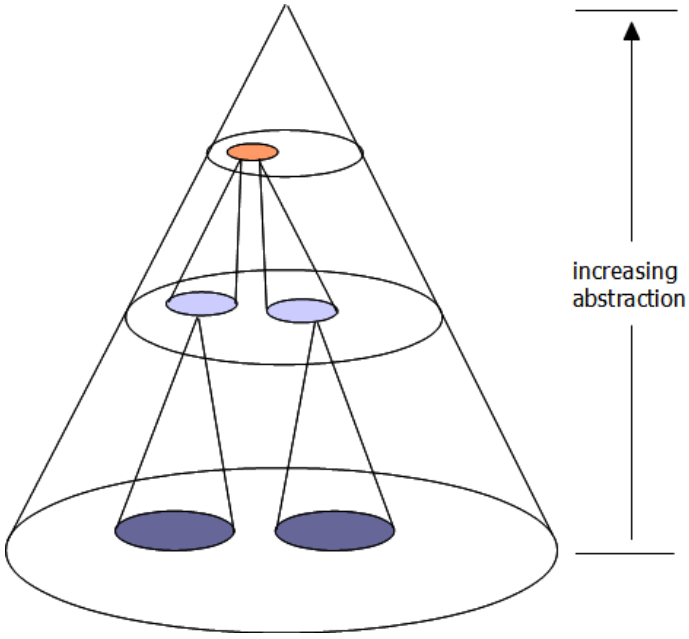


*Figure 2.8: Simulation abstraction levels*

Simulations can be classified using a number of criteria. They can be either *Deterministic* or *Stochastic*. A deterministic simulation relies on an underlying mathematical model and does not contain any elements of probability (Perros, 2009). A deterministic simulation will behave predictably, and if the same input values are retained, it will produce the same results each time it is run. A stochastic simulation involves elements of probability. Thus even if the initial conditions and inputs are retained from one simulation run to the next, it may result in different outcomes. Many simulation techniques rely on this element of probability because the simulation of real-life systems if often based on the simulation of random external events.

Additionally, there are a number or broad categories that different types of

simulations tend to fall into. While the list is not definitive it includes Monte Carlo, Continuous, and Discrete Simulations.

### 2.3.1 Monte Carlo Simulations

The term Monte Carlo simulation was first coined by John Von Neumann and Stanislaw Ulam working on the Manhattan Project. This method was named after the city in Monaco, well known for its casinos, as this simulation involves the input of random variables (or chance). In a standard deterministic simulation as depicted in Figure 2.9, the model takes in several variables and produces a specific output. Executing the simulation with the same variables will always produce the same output as indicated.

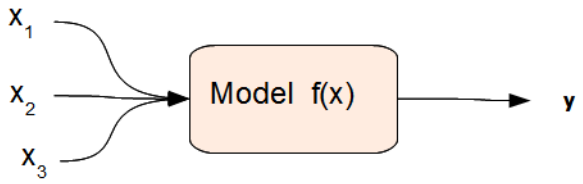


*Figure 2.9: Representation of deterministic model*

However, if the system we are modeling is stochastic (non-deterministic) in nature then the same inputs will not produce the same output on subsequent runs of the simulation. Additionally, if we have an essentially deterministic system which is inherently complex in nature, or takes in many complex parameters, then a deterministic simulation may be unrealistic due to time constraints or computational power needed. We need a different method for simulation of these types of processes.

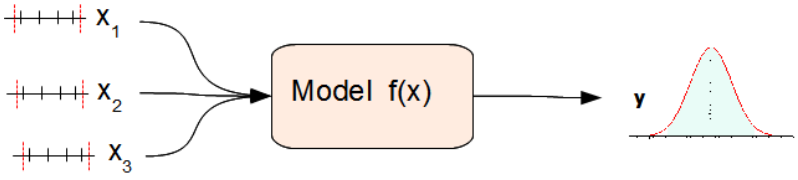


*Figure 2.10: Representation of Monte Carlo simulation model*

A Monte Carlo simulation allows us to address these problems by using statistical sampling to approximate a solution over many iterations of the simulation. As depicted in Figure 2.10, a Monte Carlo simulation takes in variables whose values are generated randomly and examines the performance of the system. When examined over many iterations of the simulation (usually measured in hundreds or thousands), the output will be a statistical distribution which we can use to estimate the performance of the system given future inputs. Thus the Monte Carlo method allows us to essentially analyze uncertainty. Such a method is extremely useful when simulating complex systems.

### 2.3.2 Continuous Simulations

Often we are concerned with systems where the state of the system is constantly changing, but time is important. We wish to observe the behavior of the system due to its changing state over time. In such cases we utilize continuous simulation techniques. Continuous simulation is concerned with modeling a system over time where the system is represented by a model whose state variables are continually changing with respect to time (Cellier and Kofman, 2006, Duivesteijn, 2006). In such cases the system is normally modeled by algebraic equations whose output varies over time (McHaney, 2009).

These simulations typically use a continuous simulation clock which is incremented in time-intervals. Depending on the nature of the simulation, we make the time intervals as fine as possible so as to capture all the relevant state changes within the system. If the time intervals are to far apart, then the simulation may miss important state changes affecting our observations of the putput.

A typical example of this type of simulation is the predator-prey problem. This simple example can be simulated by two equations with respect to time:

$$x'(t) = ax(t) – bx(t)\, y(t)$$

$$y'(t) = - cy(t) + dx(t)\, y(t)$$

Where the prey population can be given at time $t$ by $x(t)$, and the predator population can be given by $y(t)$. if $x'(t)$ represents the growth of the prey population over time, then this can be determined by assuming that the prey will grow over time by $ax(t)$ in the absence of predators (for some $a > 0$) and the death rate of the prey due to interaction with the predators will be proportional to $x(t)y(t)$, that is, $bx(t)\ y(t)$ (for some $b > 0$). Conversely, we can assume that the predators will die by the rate $cy(t)$ (for some $c > 0$) and the birth rate of the predators depends on both of the population sizes according to $dx(t)\ y(t)$ (for some $d > 0$). Thus we can use a continuous simulation to observe over time the effects of changing the parameters, $a,\ b,\ c$ and $d$ in the above equations (Duivesteijn, 2006).

### 2.3.3 Discrete Event Simulation

With Continuous simulations techniques, often the simulation can be very inefficient depending on what we are interested in. For example, there may be large periods of time within the simulation where nothing happens and no changes will be observed in the system. In these types of simulations we may not be interested primarily in what happens over time, but in what happens at the points when the system changes. In these instances we can utilize Discrete Event Simulations. Discrete Event Simulations differ in so far as we don't simulate the passage of time, but rather the processing of events which change the state of the system, as they occur within the model. Consequently in these simulations, large blocks of time may occur in between events, but the passage of time is not modeled, but rather the event arrivals and subsequent changes to the system (McHaney, 2009).

Discrete event simulations are often applied to queuing problems such as production queues, network traffic queuing and simple problems such as bank queue analysis.

## 2.4 Agent Based Modeling and Simulation

Agent Based Modeling is a relatively new and revolutionary approach to modeling specific types of systems (Bankes, 2002). Traditionally, models or simulations have sought linear solutions to problems that can be very complex. For example, the predator-prey problem discussed in section 2.3.2 models the growth rates over time of the populations of predators and prey as they interact. However the model is based on equations which model the interaction as a function of time based on input parameters. These types of models can be very sophisticated and if the interaction proceeds in a linear fashion then the models can produce predictable outcomes.

In reality, such interaction is rarely linear in nature and the linear models are unable to account for this non-linear behavior often observed. Agent-based modeling allows us to address this non-linear behavior by directly representing individuals or objects in the model and defining their interaction behavior. In an agent-based model, a system is modeled as a collection of autonomous agents, where each agent evaluates its situation periodically and will make decisions on how to act based on pre-defined rules (Bonabeau, 2002).

Agent-based models usually have the following elements in common (McHaney, 2009):

"*Multiple agents modeled and scaled with various levels of detail*

*Decision-making heuristics and rules*

*Adaptive behaviors or learning*

*Interaction rules or topology*

*Environment for interaction often consisting of constrained resources*"

The first of the elements above, "Multiple agents", implies that when discussing agent based models, we are invariably talking about Multi-agent models, and hence Multi-agent simulation.

### 2.4.1 Multi-Agent Based Simulations

The difference between multi-agent models and multi-agent simulations are analogous to the difference between Simulations and Models we discussed in Section 2.3 . A multi-agent simulation is the use of a multi-agent model to study the output of the model as the simulation progresses. Multi-agent simulations require the development of multi-agent models that try to capture the essence of complex real-world systems within the model (Ruas et al., 2011). There is an inherent relationship between complex systems and multi-agent simulations and we will explore this briefly in the following section. However multi-agent modeling and simulation is a field of research at the crossroads of simulation and distributed artificial intelligence, the latter providing an infrastructure for the modeling and understanding of the interactions between agents in the model (Ruas et al., 2011).

At the heart of multi-agent models, there are four general concepts which make them extremely useful as a research tool, especially when applied to studying any system based on human interaction (Michel et al., 2009). These are:

- *"Autonomous activity of agents"*: That is, the degree of autonomy that the agents exhibit while trying to achieve their objective. This will also include their ability to modify their own behavior as the simulation progresses in order to repress unsuccessful actions.

- *"Sociability of agents"*: This is a measure of the ability of the agents to interact socially with other agents. This centers on the fact that in a multi-agent systems, agents are not isolated, and there is a duality in that the interactions between agents form societies or groups and additionally the organizations of the groups will help define the behavior of the agents.

- *"Interaction between agents"*: This concept essentially forms the connection between the previous two. The term interaction here does not refer to actual communication between agents, but rather the effect on each agent by the interleaving of all the agents actions. During their autonomous activity, each

agent will decide on their own actions with the resultant sum of the individual actions affecting the state of the system. This in turn will influence the individual agents behavior, possibly changing their future choice of actions. Through effects of the individual actions we can see forms of cooperation or conflict arise.

- *"Situatedness of agents"*: This refers to the environment in which the agents are placed. The environment itself is an important construct as the conditions present in the environment will help determine the agents current state and how the agents interact and affect each other throughout the simulation. We can consider the environment as a common thread grouping the agents together into a common setting defined by a set of possible states.

The utility and the effectiveness of multi-agent simulations to model specific types of real-world problems lies in the underlying repetitive processes determined by the above four concepts. The agents are first placed into an environment which determines their state. From their current state an autonomous agent will determine a course of action to undertake. During their processing (or '*thinking*') cycle, the agents may communicate directly or indirectly with other agents. Once having determined the final course of action, the agent will proceed with this action, which in turn may place the agent in a new state as well as affect the current state of other agents in the simulation. As each agent follows this cyclic approach, the combination of interactions and agent actions act together which results in a complex simulation of the underlying system.

In multi-agent simulations, it is the interactions between the agents that become important to the model. In contrast to the traditional deterministic simulations, by modeling the interactions between agents at the local level we can often observe a more realistic simulation output. The interactions between agents can often lead to a modification of behavior which is not taken into account by deterministic solutions. As a result of this model, the global behavior of the system being modeled is a direct result of the local interactions between the modeled agents. This is depicted in Figure 2.11.

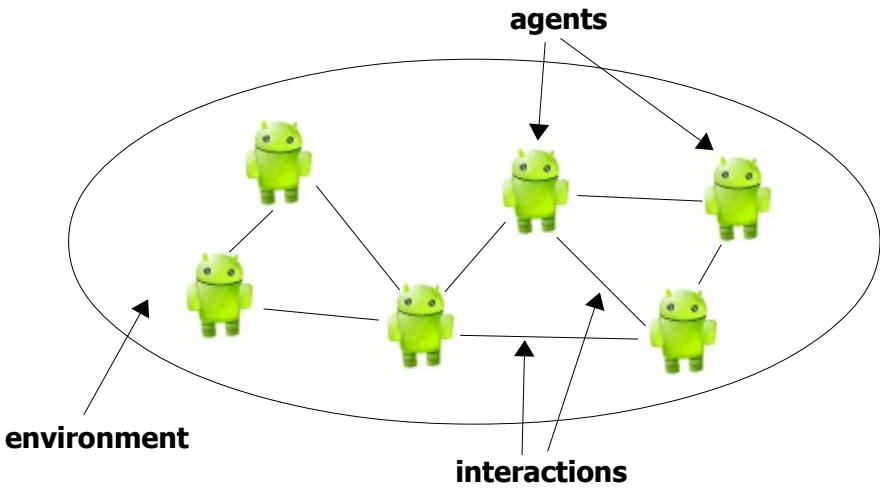


*Figure 2.11: Elements modeled in multi-agent system (adapted from (Michel et al., 2009))*

In a multi-agent system, each agent is just one member of a population, and the overall goal of an agent may be a shared goal among a subset of the population, or an individual goal. In each case, the agents goal may be best achieved through cooperation with one or more other agents.

### 2.4.2 Simulation and Complexity

It is the intrinsic nature of multi-agent systems that is driving researchers to apply them as simulation techniques, when building models of systems viewed as non-linear complex systems. The term complex system comes from the relatively new field of Complexity Science (Macal and North, 2005). Complexity science is an intersection of many diverse fields, including mathematics, AI, computing, engineering, sociology, biology, and many others. The emergence of complexity science has led us to seek different methods to simulate systems that are too complex for deterministic mathematical solutions.

What makes a system complex? The best known definition of complexity is the KCS (Kolmogorov-Chaitin-Solomonoff) definition (Smith, 1995), which places

complexity somewhere between order and randomness. However, a less rigorous definition defines a complex system as one in which an algorithm could describe the behavior, but where mathematical models do not provide efficient solutions to understand and then predict underlying phenomena (Marcenac, 1996). Put very simply, complexity theory deals with systems that have many parts that interact in non-linear ways.

Many traditional approaches to simulation and modeling have been linear in nature, that is, processes and actions are directly proportional to, or related to input. As in Newtonian science, in such systems, cause and effect are usually separate (Lucas, 2000). In many simulations this is a valid model, but in complex systems that have many components that interact, cause and effect cannot be separated, and positive non-linear feedback results. In these situations, a non-linear approach to simulation may prove more desirable.

A non-linear system is one in which the output is not proportional to the input. Or in other words, the output of the system is determined from not only the output of the components, but also from the complex interactions of the system components (Ilachinski, 1996, Bossomaier and Green, 1999). Traditional linear simulations find it difficult at best to model this behavior, but by using a MAS, the components of a non-linear system can be modeled by the agents, and the interactions between the components are modeled by the local interactions between agents.

The overall behavior of a MAS simulation caused by the local interaction of the agents is called the emergent properties (or behavior) of the simulation (Hunt and Saias, 1998).

### 2.4.3 Emergence

One of the most important ideas to come from complexity theory is the formal notion of emergence (Gilbert and Troitzsch, 2005). A system of objects displays emergence when the interactions between objects at one level produce objects of a different type at another level. Or, results in objects displaying a collective behavior

that they would not otherwise display individually.

The emergent properties of a simulation are the outcomes of behavior that we cannot reliably predict from the initial conditions. In fact the non-linearity of MAS simulations imply that when beginning a simulation with what seems like almost the same initial conditions, any difference is amplified by the agent's interactions and different emergent behavior can result. This is commonly referred to as the 'Butterfly Effect'.

Interaction between agents is central to the design of a Multi-agent System (Ferber, 1999), and is a consequence of their plural nature. Interaction occurs when two or more agents interact through a series of events during which the agents are in contact. We normally think of interaction in terms of human communication (as in speech), which is usually modeled as messaging passing between the agents, but interaction may take on other forms. If interaction is a consequence of the nature of MASs, then we should ask the question, "Do we always get quantifiable emergent behavior as consequence of the interaction"?

The answer depends on the patterns of interaction within the MAS. Green (Green, 1993), discusses this in terms of connectivity in Graphs. That is, we can model the interaction between agents as a directed graph, where our directed graph is a pair G = <V, E>. V represents a list of nodes in the graph, and E represents a set of 'edges' connecting two elements of V. In a MAS, V would represent the agents in the simulation whereas E represents the current interaction between the agents. For a MAS, G changes over time as V and E change. If over time, G displays relatively low connectivity and only small fluctuations in connected sub-graphs, the MAS can be considered in a steady (or freeze) state, and is unlikely to produce emergent behavior. Also if the changes in G are such that it displays very high connectivity, and E displays randomness from one state of G to another, the MAS can be considered chaotic, and again unlikely to display emergent behavior.

Emergent behavior is likely to occur in models where the changes in G are such that G displays near critical connectivity, and changes in E display only local

fluctuations. This point in-between a steady state and chaos (instability with order), is called the 'Edge of Chaos' (Lucas, 2000). Once emergent behavior becomes evident, we need to know whether a particular emergent behavior is a necessary consequence of the system or just a contingent one.

There are a number of ways that we can analyze and understand emergent behavior of a MAS, but broadly speaking they fall into two overall categories, design techniques, and inspection. Good design techniques based on well formed models of behavior help us understand the behavior patterns of individual agents. However such techniques cannot foresee all types of emergent behavior as even an MAS with simple rules can generate very complex emergent behavior which can be difficult to understand and predict. Analyzing such behavior is usually done via either a detailed scenario analysis, or by a Monte Carlo approach. Terán, Edmonds and Wallis (Terán et al., 2000), describes a constraint-based architecture approach for searching possible models of a MAS and testing the necessity of certain emergent behavior over a range of system parameters.

Emergent behavior in a MAS is particularly important because of its potential use as a tool for discovery and formalism (Gilbert and Troitzsch, 2005) The use of MASs for simulation and study of social phenomenon relies heavily on emergent behavior to test theories in the underlying domain. Simulation of battlefield behavior, is basically a social simulation, where the environment is hostile for agents inhabiting it.

## 2.5 Combat Simulation

Military organizations exploit simulation techniques extensively and in fact simulation has its roots in military history. The Australian Army utilizes approximately 70 live, virtual and constructive simulations (Colton, 2001). These simulations are layered in a hierarchy from the conceptual to physical levels, with conceptual simulations used for overall insights into scenarios, and physical simulations utilized to concentrate on the results (Barlow and Easton, 2002). With the recognition of land

combat as a complex adaptive system, simulations at the conceptual level using multi-agent techniques, called distillations, have of late, received much attention (Brandstein, 2001).

Distillations represent a move away from the traditional constructive combat simulations, which are usually based on linear models of attrition. Such simulations often must embody the details of what they are trying to show, whereas the agent-based approaches attempt to produce the macro-level behavior by micro-level interactions (Goldspink, 2002). At the conceptual level, these simulations provide insights into land combat by allowing the strategist to specify the local interaction between the combatants and then observe the emergent behavior of the system. There are several of these distillations currently used by the Australian, New-Zealand and U.S. army, and these include, ISAAC, EINSTEIN, MANA and Socrates (Barlow and Easton, 2002).

There have been many simulation models of combat environments, but traditionally these models have sought linear solutions. For example, the "Lanchester Mechanism" uses coupled differential equations to model attrition rates in combat (Beckerman, 1999). The equations:

$$dB/dt = kr\ R(t) \quad \text{and} \quad dR/dt = kb\ B(t)$$

model constant kill rates $k_r$ and $k_b$ for Red and Blue teams whose weapons stock $R$ and $B$ respectively is a function of time $t$. In this linear solution, increasing the weapons stock of a particular team, increases the kill rate linearly.

Combat may be modeled by linear simulations, and if both teams proceed in a linear fashion then such simulations may produce predictable outcomes. However, if an opposing side is capable of adapting to the combat efforts of the other, then the resulting combat is unlikely to be linear. In fact, military conflicts (particularly land combat) possess almost all the main features of complex adaptive systems (Ilachinski, 1996).

They are composed of a large number on non-linearly interacting parts; these parts are usually organized into a hierarchical structure; action generally proceeds with decentralized control; there is a high degree of self-organization; and in order to survive the parts must adapt to the changing environment.

SunTzu (SunTzu, 2005) , described combat with an analogy to that of fluid dynamics, which is also an appropriate comparison of combat to a complex system. So in simulating combat on the battlefield, it is appropriate to use non-linear models. In particular, Multi-Agent Systems provide us with a means to simulate individual combatants on the battlefield, and the non-linear local interaction between the combatants by communication between respective agents. With such a simulation, the emergent behavior then becomes an important consequence of the model.

There are a number of Multi-Agent System simulations of land combat currently in use or under development. ISAAC (Irreducible Semi-Autonomous Adaptive Combat), is an agent based model used by the US Marine Corps Combat Development Command. ISAAC was built as a "proof of concept" system (Ilachinski, 1999) , and is based loosely on mobile Cellular Automata rules. Each agent in ISAAC (called an ISAACA) is equipped with four main characteristics, Doctrine, Mission, Situational Awareness and Adaptability. Each ISAACA is also given a personality, which is defined by a six element weighted vector. It is the personality that each of the ISAACAs has, which help determine its action at each time instance in the simulation (Ilachinski, 1997).

EINSTein (Enhanced ISAAC Neural Simulation Toolkit) is a follow-on improvement to ISAAC. It is agent based (as ISAAC) and among its enhancements include a GUI interface, Object oriented code base, genetic algorithm and neural net reinforcement learning, and pattern recognition toolkits. Both ISAAC and EINSTein include command structures (or hierarchy of information levels) "hardwired" into the code (Ilachinski, 1999) .

SWarrior is an agent based simulation, based on ISAACs agents, but utilizing the SWARM modeling environment (Hunt and Saias, 1998). SWARM was developed at

the Santa Fe Institute and the basic architecture of Swarm is the simulation of collections of concurrently interacting agents. That is, it provides a simulation environment for agents independently specified, to run in. The goal of SWarrior is to enable the adaptation of SWARM into an analytical tool that uses multiple run agent based simulations. The aim is to help provide insights into future operations.

Hunt (Hunt and Saias, 1998), discusses ACME (Adaptive Collection Management System). This is an agent based simulation that models a mobile enemy command post. The aim is to use this simulation to aid in the development of a temporal information management system on battlefield positioning. This is especially important in a mobile situation when the map of the battlefield degrades over time. While this is not a battlefield simulation as in the previous examples, it does contain elements of a combat situation.

With all the current Modeling and Simulation, is there room for yet another simulation? It is only through a diversity of simulation techniques and experimentation that progress and understanding can proceed. The application of multi-agent simulations to combat simulation is still in the early stages, and there is still much we don't know. This is particularly true with emergent behavior from such simulations.

At the heart of these military distillations are the agent control paradigms used to control the behavior of the agents during the simulation. These paradigms effectuate "low-level" observable behavior, or human control strategies, by implementing a blend of action and reaction skills (Henninger et al., 2001). These skills are effected by weighted factors that represent various attributes such as attraction, repulsion, aggression, cooperation etc… and ultimately determine the action an agent will take (Barlow and Easton, 2002). Agents controlled by these human control strategies represent highly instinctual entities, acting quickly based on the current local dynamics of the environment. Cognitive behavior is not directly observable in these simulations but can be inferred from the low-level behaviors.

The decision making process on the battlefield is complex, and while soldiers react to situations based on years of training and experience, decisions are not made

within a cognitive void. Agents that are capable of learning in these simulations may provide a more realistic basis for representing individuals in multi-agent system (Gugel et al., 2001b, Sun, 2001). One of the significant characteristics of learning agents that is motivating their use in simulations, is their ability to adapt in a complex dynamic environment (Vaughan and Connell, 2000). However, Petty (Petty, 2001) questions the use of learning paradigms in Computer Generated Forces, arguing that many such systems produce unrealistic human behavior or perhaps improved behavior but resulting in confounded results.

Provided that the distillations facilitate realistic environments based on the physics of the world they are simulating, agents' resulting actions should be sufficiently restricted to plausible human behavior. The results of a learning experiment involving learning agents may not be reproducible. However, that is also the nature of what we are dealing with in a complex adaptive system.

## 2.6 Learning Agents in Simulations

When considering the problem of incorporating learning techniques into agents in a multi-agent simulation, we have a number of Artificial Intelligence (AI) techniques to chose from. However, it is essential we chose a technique applicable to the multi-agent framework. That is, we have agents situated in an environment which interact, and learn as the simulation unfolds. Being an essentially complex system, small fluctuations in the initial conditions or even the interactions between agents can lead to vastly different outcomes, and hence the progression of states that agents can be in as the simulation progresses. This can have an effect on the choice of learning techniques.

Concept learning is used to induce general functions from specific training examples (Mitchell, 1997a). While this learning techniques would allow an agent to search through a set of training examples looking for a best fit in deciding a course of action, the number of training examples required to effectively work in a complex system may be prohibitive. Bayesian learning takes a probabilistic approach to deciding optimal courses of actions. However, the assumption underlying the use of Bayesian learning is that the choices of actions which are effective would be based probability

distributions (Mitchell, 1997a). Again, given that the underlying simulations is complex, we could not guarantee such distributions. In fact, any evident distributions are liable to change as the simulation progresses.

Other possible choices for learning techniques include a Genetic Algorithm approach, or the use of Artificial Neural Networks. While Artificial Neural Networks are promising, Reinforcement Learning is a technique particularly suited to, and often utilized when we have autonomous agents that sense and act in their environment, and learn by interaction with the environment (Mitchell, 1997b).

### 2.6.1 Reinforcement learning

Distillations create a virtual environment where the agents move around and interact with each other and the environment, and thus are good candidates for the application of reinforcement learning techniques. In reinforcement learning, an agent perceives the world as a series of states, and then chooses actions based on those states. In the next time frame, the agent receives a reward from the environment, and updates an action value function based on the reward, which will affect its choice of actions in future steps. The goal of the agent is then to maximize the overall reward.

There are a number of elements required in a Reinforcement Learning approach. These are, the *agent*, the *environment* (being the main elements) along with a *policy*, a *reward function*, a *value function* and possibly a *model of the environment*, depending on the reinforcement learning approach. The agent is situated within the environment and perceives itself as being within one of a number of possible states. The policy determines the agents' behavior as it essentially maps from a set of states to a set of possible actions. The reward function essentially defines the goal of the agent as it maps the combination of the current state and chosen action to a reward. The agents overriding mission is to maximize the reward. Thus, over time the agent learns the optimal action to take given its current state (Sutton and Barto, 1998).

The traditional reinforcement-learning update cycle is shown in Figure 2.12. This illustrates that at time $t$, an agent will sense it's current state $S_t$ at time $t$, and choose

action $a_t$ at time $t$ to maximize its reward $r$. This will result in a possibly new state $S_{t+1}$ and time time $t+1$, and a corresponding reward $r_{t+1}$, which then feeds back into the agent so it can new choose a new action based on the new state.

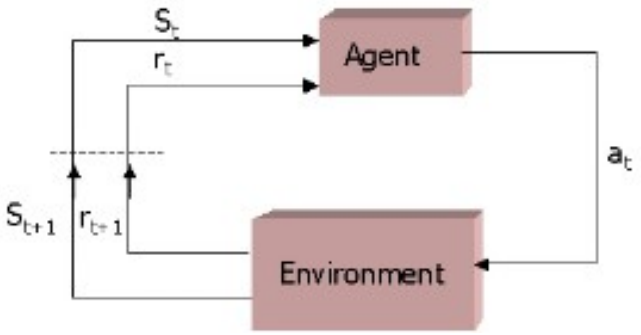


*Figure 2.12: Traditional reinforcement learning update cycle (Sutton and Barto, 1998)*

One of the main differences of reinforcement learning is that it is essentially *unsupervised learning*. Most learning techniques are classed as *supervised learning*, in that the learning takes place from examples provided during a training phase. While this is suitable for many types of applications, in problems involving interaction between the agent and the environment, often satisfactory training cannot be provided. The agent must learn by experience and thus the reward function is crucial in defining the agents goals. This additionally highlights another major difference between reinforcement learning and other types of learning. In reinforcement learning, the objective is to maximize the reward function by choosing actions which the agent has already discovered to yield a high reward. However, in order to find these actions it has to try actions it may not yet have tried. Thus the agent must balance actions between '*exploration and exploitation*' (Sutton and Barto, 1998). In exploration the agent will randomly select an action regardless of reward considerations in the hope of finding a new high reward action. An agent cannot exclusively follow an exploration or exploitation policy and be successful, it must use both. Thus the action selection policy is normally one of 3 common policies:

- $\epsilon$ - greedy – most of the time the action returning the possible reward will be chosen. Occasionally with small probability $\epsilon$ an action will be selected uniformly at random regardless of the reward.

- $\epsilon$ - soft – this is similar to the $\epsilon$ -greedy policy, with the optimal action selected with probability (1- $\epsilon$ ) and a random action being chosen uniformly at other times.

- Softmax – with the softmax option, when a random action is chosen it is done so on a fuzzy basis with regards to a weighting assigned to each action. Thus the worst actions are less likely to be chosen.

While there are a number of different reinforcement learning techniques, Sutton and Barto (Sutton and Barto, 1998) classify these into three broad categories: Dynamic Programming techniques; Monte Carlo Methods; Temporal Difference learning. Dynamic programming techniques describe a number of reinforcement algorithms that find optimal policies but require a perfect model of the environment. In many situations this can be unrealistic, particularly if we are modeling an agent exploring an unknown environment, or we are modeling a battlefield scenario where the '*fog of war*' prevails. The Monte Carlo methods are episodic in that they solve the reinforcement learning problem by letting the agent learn from experience gained by averaging the performance based on a number of episodes. That is, in a reinforcement learning simulation, the agent learns at the end of the simulation-run (or episode) by examining the rewards received throughout the episode as a result of the actions. A number of episodes are then utilized to let the agent average the returns. However, the purpose of a simulation is to provide an abstraction of a real system, thus the ultimate goal is to mimic reality as closely as possible. In many situations, agents will learn as the simulation unfolds. This suggests a non-episodic learning approach where the agents learn from direct experience as the simulation progresses.

Temporal Difference Learning algorithms allow an agent to learn from direct experience without having to possess a model of the environment. The agent does not need to wait for the end of an episode and can update its estimates of reward based in

part on its previous estimates (Sutton and Barto, 1998). Such algorithms best mimic the learning which occurs when we are modeling social environments where humans learn by interacting with the environment, and other humans without the luxury of repeating an episode if they fail at the first. While there are a number of temporal difference learning algorithms, Q-Learning represents an off-policy control algorithm, relatively easy to implement, and this algorithm is often used to incorporate intelligent behavior into agents (Gugel et al., 2001b, Gugel et al., 2001a, Lapin et al., 2001, Tan, 1993).

### 2.6.2 Q-Learning

The development of the Q-learning algorithm was an important milestone for reinforcement learning (Sutton and Barto, 1998). The Q-Learning algorithm was developed by Watkins (Watkins, 1989) and is considered an *off-policy* control algorithm. This means that the Q-Learning algorithm can update the estimated value functions for those actions that may not have been tried. That is, off-policy algorithms can distinguish between exploration and exploitation whereas *on-policy* algorithms cannot. The action selection policy of the Q-Learning algorithm is relatively easy to implement.

The equation representing the basic *one-step Q-Learning* is given below in Equation (2.1) (Sutton and Barto, 1998). In this equation, an agent will receive a reward, *r*, at time *t+1*, for choosing action *a* when in state *s* at time *t*. Thus the agent learns at each time step by processing this information and updating its own action-value matrix.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \qquad (2.1)$$

In Equation (2.1), $Q(S_t, a_t)$ represents the current estimate of the action-value pair, action *a* in state *s* at time *t*. This is updated from the current value of $Q(S_t, a_t)$, the reward $r_{t+1}$ received from taking action *a* while in state *s* at time *t*, (received at time

*t+1*), and the maximum action-value (over all possible actions) of the resultant state at time *t+1*. The values $\alpha$ and $\gamma$ represents the values of the step-size, and discount factors respectively.

The step-size is a small fraction that influences the rate of learning, and if constant, allows us to effectively track the optimal action-value function *Q** as a non-stationary problem. The discount factor determines the present value of future rewards and lies in the range $0 < \gamma < 1$. If $\gamma$ is set at zero (0), then the agent is only concerned with maximizing the immediate reward, and as it approaches one (1), the agent becomes more prudent and takes into account future rewards.

The algorithm implementing Q-Learning is given below in Listing 2.1:

Initialize $Q(s,a)$ arbitrarily
Initialize $s$
Repeat for each step of the episode
    Choose $a$ from $s$ using policy derived from (eg $\epsilon$ -greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$
    $s \leftarrow s'$
until $s$ is terminal

*Listing 2.1: Q-Learning algorithm (Sutton and Barto, 1998)*

### 2.6.3 Multi-Agent Reinforcement Learning

The application of learning techniques applied to multi-agent systems is a growing field (Claus and Boutilier, 1998), and an exciting, emerging area of research is Multi-agent Reinforcement Learning. Multi-agent reinforcement learning is concerned with how an agent can learn to act optimally in an unknown environment through trial and error interaction, and in the presence of other adaptable agents (Khoussainov, 2004). Traditionally, reinforcement learning is a technique particularly suited to, and often utilized, when we have autonomous agents that sense and act in their environment,

and learn by interaction with the environment (Mitchell, 1997b). However, as indicated, the formal theory of Reinforcement Learning is based around a state-space conforming to a Markov assumption, and thus limited to a single agent. The practical applications of reinforcement learning utilizing a single agent are generally specialized and limited in nature, particularly with recent trends in AI research towards Agent Based Systems.

A multi-agent system, while generally Markov-like in nature is essentially a non-Markov decision process, as the interaction between the agents themselves and concurrent learning affect the state transition probabilities (Arai et al., 2000). Shoham, Powers and Grenager (Shoham et al., 2003), argue that utilizing reinforcement learning for multi-agent learning systems is fundamentally flawed due to the uncertain nature of the problems being tackled. One of the measurements of performance often used for reinforcement learning algorithms is convergence. Yet, in multi-agent reinforcement learning, convergence is not guaranteed. Bowling and Veloso (Bowling and Veloso, 2002), also discuss the two desirable properties for multi-agent learning systems, rationality and convergence. They show that in multi-agent systems using reinforcement learning, while rationality exists, generally convergence does not. The normal notion of an optimal policy does not exist, since at any time during a learning trial the optimal policy will depend on the policies of other agents.

Under such conditions, the optimal policy becomes a moving target, and the reward function can return a reward to the agent which seems arbitrary in nature. This is a particular difficulty for temporal difference algorithms that assume that past rewards are a prediction of future rewards given the same state/ action combination (Bowling and Veloso, 2002). Notwithstanding such arguments, a number of researchers have attempted to formalize a theoretical foundation for multi-agent reinforcement learning.

Littman (Littman, 1994), explores the use of stochastic games as formal framework for reasoning about multi-agent systems. In particular, using zero-sum stochastic games, Litmann was able to show the selection of an optimal policy using a Q-learning like algorithm. This work was extended by Hu and Wellman (Hu and Wellman, 1998), where a broader framework of general-sum stochastic games was used as an underlying framework. In such games we must drop the concept of 'optimality' for

that of 'best response'. In a modified Q-learning algorithm, Hu and Wellman managed to show convergence under certain conditions, however, only with infinite trials.

Other researchers have also explored the use of stochastic games as a formal framework for multi-agent reinforcement learning (Bowling and Veloso, 2000, Chalkiadakis, 2003, Hansen et al., 2004, Khoussainov, 2004). However, interestingly, Bowling and Veloso address the convergence problem for multi-agent reinforcement learning with a variable learning rate (Bowling and Veloso, 2002). Using the WOLF (Win or Learn Fast) principle, the learning rate is increased when the agents are performing poorly and slowed when they are performing well. By varying the learning during play, convergence can be achieved, but again under certain conditions.

Despite the problems of the applicability of a formal framework to multi-agent reinforcement learning, there have been many reported successes in utilizing this technique. One of the earlier reported successes was Tan (Tan, 1993). Tan investigated a limited hunter-prey problem with 2 hunters controlled by a Q-learning algorithm. Specifically, Tan experimented by allowing the hunters to communicate sensations, policies and episodes. In these trials, the hunters learned a form of cooperative behavior, and did better in each case than the independent reinforcement learners. Although these experiments were limited in the number of agents utilizes, two hunters at most, similar promising results have been obtained by other researchers utilizing many more learning agents.

Nunes and Oliveira (Nunes and Oliveira, 2003), utilizes a traffic control simulation with 5 learning agents, where the agents learn to control traffic flow through intersections. The agents learn from each other by exchanging advice, which is a selective episodic exchange. Although the learning agents utilized different learning algorithms, including Q-learning, all did better than their independent learning counterparts when cooperating with the other learning agents. Mataric (Mataric, 1998) also describes an experiment where four Q-learning agents share sensory and reinforcement information to learn social rules. These agents all have the same objective, to collect pucks in a limited state space, but the effect of the communication is to increase the scope of impact of individual agents, thus improving their

performance.

Versino and Gambardella (Versino and Gambardella, 1997), discuss experiments with teams of up to 14 Ibots (Integrating Robots). These Ibots are governed by a reinforcement learning algorithm, and work together to discover a team solution to artificial tasks created to study the degree of integration possible. The success of the Ibots in learning team solutions is a little different to previous research in that no communication exists between the Ibots. However, the lack of communication is really a lack of explicit communication, as the learning of the Ibots is governed by a reinforcement signal that evaluates team performance, not individual performance. Thus the Ibots tend to lose individuality and behave as sub-parts of a whole. Other forms of implicit communication between agents can be discovered in further research, for example (Price and Boutilier, 1999, Sen et al., 1994). Price and Boutilier (Price and Boutilier, 1999) utilize implicit communication in reinforcement learning agents by having agents observe the actions of more experienced mentor agents. The effect here is that the observer agents converge, or '*learn*' more quickly than control agents utilizing their own individual strategies.

The success of the agents in the previous applications of reinforcement learning to multi-agent systems lies in the communication between the agents. This communication can be either explicit or implicit, but the effect is to accelerate the learning rate of the agents involved. In reinforcement learning, this would also lead to a variable learning rate depending on the learning events taking place at any particular time throughout a simulation. The variable rate was utilized in (Bowling and Veloso, 2002) above to address convergence. Given the apparent success of reinforcement learning in many multi-agent simulations, one question worth considering is whether optimal convergence should always be sought at all costs (Chalkiadakis, 2003).

## 2.7 Cooperative Learning in Multi-agent Systems

In a multi-agent system utilizing Q-Learning, each of the agents are individual learners, and each learns from interaction with the environment, and other agents. They may be part of a group whose learning is directed by the same reward function, but each agent will learn independently from the others. If we have a group of such agents, they have an implicit goal in common by virtue of the reward function. Norman (Doran et al., 1997) argues that sharing the same implicit (or explicit) goal is not enough for cooperation to exist, and that there must be intent to cooperate. However, in contrast, Doran and Franklin (Doran et al., 1997) do not require intention to cooperate but state that cooperation exists when the action of the agents satisfy both or either of the following: the agents have a common goal and their actions tend to achieve that goal; the agents perform actions to achieve goals and also the goals of other agents. In such a situation, the cooperation is an emergent behavior

In an environment which is hostile, such emergent cooperation alone may not be sufficient for the group of agents to achieve their common goal. An agent, being an individual learner, simply may not learn fast enough to function correctly within its environment before it is eliminated by either the environment, or other agents. In order to combat this, we can implement a more explicit form of cooperation between the agents in achieving their goal, or we could allow the agents to cooperate in the learning process. As evident in the literature, there are a number of ways to achieve cooperation, but if the agents cooperate in the learning process, the cooperation on achieving the implicit common goal will still be an emergent behavior. Nonetheless, accelerated learning can improve the task performance of the individuals.

There are a number of reasons why we might want our agents to cooperate with each other during the learning process, and these include scalability, speed fault tolerance and encapsulation (Sian, 1991). One of the criticisms of incorporating intelligent agents into military simulations by (Petty, 2001), was that often the systems were not scalable. Using a multi-agent system where the intelligent agents cooperate in learning could reduce these concerns. In a multi-agent system where the environment is hostile to the agents, the speed of the learning process becomes particularly interesting..

A group of individual Q-Learning agents placed in a combat scenario against a better trained team of non-learning agents, utilizing a human control strategy, cannot learn fast enough to survive. If the agents in a multi-agent system can cooperate in the learning process, we can effectively parallelize the learning and achieve an increase in the speed of learning (Darbyshire and Wang, 2003).

Dayan and Hinton discuss speeding up the reinforcement learning process using a multiple resolution approach (Dayan and Hinton, 1993). In this approach, a managerial hierarchy is set up using reinforcement learning where high-level agent managers learn how to sub-delegate tasks to others. However, in this research, we don't want to specifically set up such a hierarchy. This is a type of emergent behavior we would ideally like to see emerge as a natural consequence of the simulation. Wolpert and Tumer discuss the emergence of Collective Intelligence multi-agent systems (COINS) (Wolpert and Tumer, 2000). In these multi-agent systems, every agent runs its own reinforcement-learning algorithm, and the questions revolve around the reward function best suited to achieve high reward gains. While these are closely related to the approach in this paper, COINS are typically viewed as a control problem for real world distributed applications.

Very closely related work to this research is presented in Tan (Tan, 1993), however there are some significant differences. Tan considered the limited hunter/prey problem in a small 10 x 10 grid world with a maximum of two hunters and two prey. This research utilizes a distillation with a 500 x 500 world with two teams of 15 agents. Tan also considered the learning problem as an episodic learning problem, and at the end of each episode, an agent communicated its entire episode-sequence to the other agent. This research considers the learning task as non-episodic, and communications occur at each time step throughout the simulation. The agents within this research are also placed within a hostile environment where they can be killed if the learning rate is insufficient.

Although the literature demonstrates that agents which communicate generally fare better than those that don't, what is generally missing is an analysis of the observable effects of the communication on simulation metrics. A computational

system, such as a multi-agent simulation, are traditionally evaluated based on the final results (Bryson et al., 2001). Thus, in multi-agent systems where communication is utilized to increase performance, we can measure the observable difference in the success or otherwise of a group of agents using communication to enhance learning. However, while the communication may lead to positive final outcomes, there must be some measurable elements of the simulation that are affected by the communication which then influence the final outcome. It is useful to identify these in order to further evaluate the results of the communication in the simulation environment.

We know from the literature that allowing agents in a multi-agent system to communicate during the learning process will increase the task performance of the group as a whole. Additionally we know from many of the research papers that communication in a multi-agent systems helps accelerate the learning process, for example (Darbyshire and Wang, 2003, Kelly and Keating, 1998). Thus we know the cause-effect relationship, but what still seems unsatisfactorily documented in the literature is a pinpointing of where the changes due to communication specifically effect either the agents or the agent system as a whole. Does the communication affect metrics of the agent system which accounts for the increased performance, or does this actually have an effect on the agents themselves, or both? The exact mechanics of the communication effects need some investigation. A detailed knowledge of these effects on the agents themselves in such an environment might help to provide avenues for further exploration of utilizing communication for group learning. It should be noted that a large body of literature exists for the research area of Competitive Agent. However, the focus of this research is solely on communication between cooperative agents, hence we have not included competitive agent literature in this thesis.

## 2.8 Neural Networks

Although this research was not directly concerned initially with Artificial Neural Networks, discussions with a previous research supervisor[1] led to a secondary interest in artificial neural networks as a model for the communication between agents in a complex adaptive system. Thus, this section provides a brief background to the study of

1 Associate Professor Dianhui Wang, La Trobe University, Melbourne, Victoria, Australia

artificial neural networks which are then used in Chapter 6, to show how the communication discussed in Chapter 5 can be modeled. While this does not represent the focus of this research, artificial neural networks do provide us with a method to conveniently and succinctly represent the communication between a group of agents in an adaptive system.

Neural Networks were inspired by the study of the human brain and its complex structure. The human brain has long been looked at as having many of the desirable computational characteristics not currently present in modern computers (Jain et al., 1996). These include massive parallelism, an ability to learn, being adaptable, the ability to generalize, and more (Cheung and Cannons, 2002). Artificial Neural Networks are named after their biological counterparts and try to model the information processing capability of the human brain. A biological neural network consists of a massively parallel system of connected *Neurons*, and understanding the structure of the neuron gives us the ability to model its essential properties.

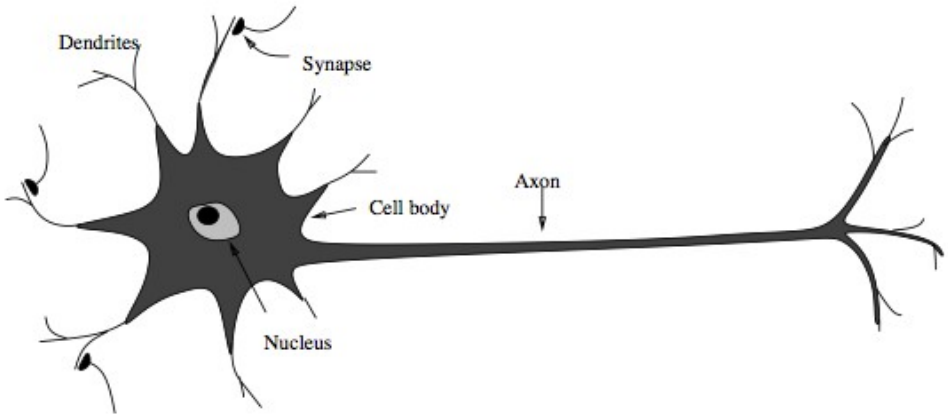


*Figure 2.13: Biological neuron (Jain et al., 1996)*

Figure 2.13 shows an abstract sketch of a biological neuron. The neuron consists of a central cell component, dendrites which are connected to other neurons via small gateway type constructs called synapses, and a longer axon which eventually branches into a number of outlying tree-like branches that then connect, via synapses, to the

dendrites of other neurons. In biological neurons, there is a flow of information through the synapses to the neurons dendrites to the cell body (soma), and then a signal is sent from the cell body along the axon to other neurons. Each of these neurons is connected in this way to between $10^3$ to $10^4$ other neurons. This general abstract structure of the brain provides us with a biological paradigm we can model in order to mimic some of the capabilities of brain itself (Rojas, 1996).

In artificial neural networks we model the biological neuron with an abstract artificial neuron, see Figure 2.14. There are many depictions of abstract neurons, but the one shown in Figure 2.14 encompasses the features of most (Rojas, 1996, Fyfe, 1996). The neuron depicted, consists of a central neuron body, a number of inputs with associated weights, and a single output.

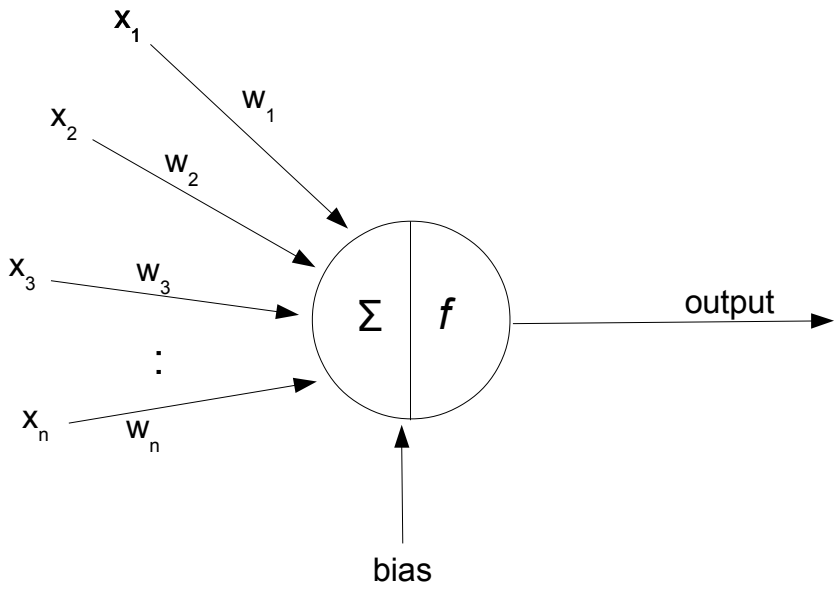


*Figure 2.14: General form of an abstract artificial neuron (Rojas, 1996)*

The inputs, $x_1, x_2, x_3, \ldots, x_n$ are the input signals to the neuron and there is an

associated set of weights $w_1$, $w_2$, $w_3$, …, $w_n$. Additionally, a bias signal is also often included as input to the neuron. The body of the neuron typically computes a weighted sum of the input signals and applies a function *f,* to produce an output *y*. This is shown in Equation 2.2

$$y = f\left(\sum_{i=1\ldots n} x_i w_i + bias\right) \tag{2.2}$$

The function *f* () is called the activation function and different models of artificial neurons differ mainly in the form of the activation function (Rojas, 1996) and the interconnection of the artificial neurons. While the activation functions are generally not restricted, often the form of the activation function utilized is a nondecreasing function in the form of either a threshold (*sgn*) function, a semi-linear function or a sigmoid function (Krose and van der Smagt, 1996). These functions a depicted in Figure 2.15. The majority of neural network applications use sigmoid activation functions (Cheung and Cannons, 2002), which have a bounded range, but never quite reach the maximum or minimum (typically 1 and -1)

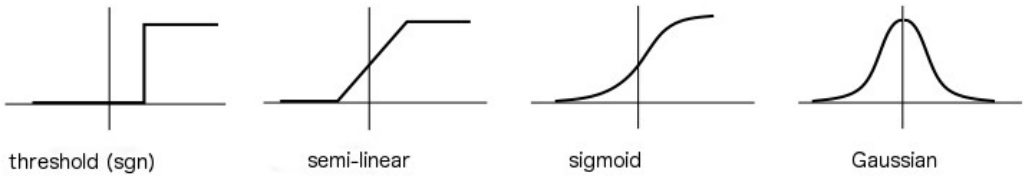


*Figure 2.15: Form of different activation functions (Jain et al., 1996)*

A threshold or *sgn* activation function is one of the simplest types of activation functions and either returns a zero (0) or a positive number, typically one (1) as the output. This effectively either turns the output of the neuron on or off depending on the weighted sum of the inputs. Such an activation function was used in one of the earliest attempts at building a learning based system based on simple structured neurons and was motivated by the early work of McCulloch and Pitts (McCulloch and Pitts, 1943). The simple neuron proposed by McCulloch and Pitts was termed a perceptron and is shown Figure 2.16.

The weights attached to the input signals are the most important element in the neural network when determining its overall behavior (Cheung and Cannons, 2002). The ability of a neural network to learn is achieved by the network iteratively updating the value of the weights on the input signal over time (Jain et al., 1996). By modifying the weights over time the network adapts, and slowly begins to better approximate the required function. When a neural network is first set up, the input signal weights can be initialized based on *a priori* knowledge, or the network can be *trained*. Training involves supplying the network with example inputs, and in *Supervised training*, also supply the outputs. In either case, the network adjusts the weights over time to achieve better results.

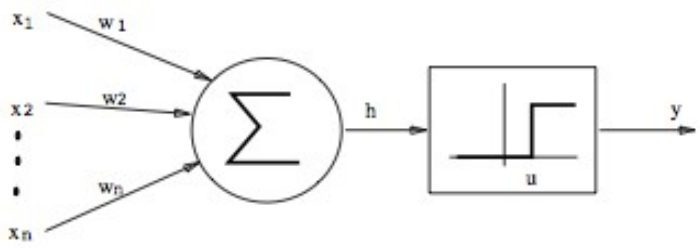


*Figure 2.16: McCulloch-Pitts early model of a neuron (Jain et al., 1996)*

While neural networks are excellent for solving particular types of problems, such as pattern recognition, feature extraction, noise reduction and prediction. Using a combination of Neural Networks and Reinforcement Learning we can explore larger-scale problems with a complexity seen more in real-world systems.

### 2.8.1 Neural Networks and Reinforcement Learning

Initial problems for reinforcement learning lay in what was termed the temporal credit assignment problem which was difficult to handle. This limited the application of reinforcement learning to smaller problems. However with the development of the class of methods called Temporal Difference learning methods (Sutton and Barto 1998) (see Section *2.6.2 QLearning*) which allowed an agent to learn successful actions based on

the difference between temporally successive predictions (Tesauro 1995). One of the temporal difference was TD(lambda) was also proposed for training a multilayer neural network (Sutton 1988). Tesauro (1988) designed TD-Gammon, a system designed to learn to play backgammon as a way of exploring the capability of a multilayer neural network trained by the temporal difference method TD(Lambda). What was surprising in the results was the amount of learning that took place given no initial training. By combining the two approaches the results surpasses the alternative approach of a supervised neural network approach. A further variation of the Tesauro work was undertaken by Papahristou and Refanidis ( Papahristou & Refanidis 2011) who also used a neural network trained by the TD(Lambda) temporal difference method to learn variations of the backgammon rules. This work showed that neural networks combined with reinforcement learning are capable of producing high performance learning algorithms for game playing.

In reinforcement learning, the reinforcement function and the states depend on the previous states and reinforcement history. When combined with a recurrent neural network, the outputs will depend on the past states and a current state internal to the system (which depends on the historical record) (Jang et.al 1997). This type of neural network might be able to learn appropriate actions by including a specific amount of past history. Additionally, reinforcement learning which involves updating a table containing values for state-action pairs and fFor large scale problems with very large state spaces this becomes impractical. We ned to use generalization techniques to deal with the large number of states. Feedforward neural networks combined with reinforcement learning are a way of dealing with the large state spaces as the function approximators in neural networks gives us this generalization capability (Coulom 2002).

There is a large body of evolving work in using neural networks and reinforcement learning in the field of robotics, particularly for visual control (Milijkovic et.al 2013) (Huang et.al 2005) as examples. Visual control of a robot requires an extremely large state space in order to provide the agent with the information it needs to learn to navigate and avoid obstacles over time (Huang et.al 2005). Traditional reinforcement learning techniques could not scale-up to make this a practical solution without generalization techniques.

The research presented in this thesis is concerned with learning by reinforcement. In particular, this research studies the effects of communication on the learning when teams of agents are working together to solve a problem. However, the treatment of neural networks in this section is provided solely as background to Chapter 6 which looks briefly at trying to model the communication between a group of agents in a complex adaptive system. The model of the neuron in Figure 2.14 and the Equation presented in Equation 2.2 provides an excellent basis for a succinct model of such communication. This will be discussed more fully in Chapter 6 Modeling Communication.

# 3

# METHODOLOGY

*The method of science is tried and true. It is not perfect, it's just the best we have. And to abandon it, with it's skeptical protocols, is the pathway to a dark age.*

– *Carl Sagan*

When any research is undertaken, it is important to conduct the research in a manner that is consistent and sound with the type of problem being investigated, and is reproducible so that verification can take place. Thus the methodology used to conduct the research is a vital part of the research process, and needs to be documented in detail to the satisfaction of not only the researcher, but also to those reviewing the research. The purpose of this chapter is to discuss and document in detail the the conceptual framework and research methodology used to gather the data used in investigating the research problem. In this chapter we look at the research methodology used at the broad level, and in the following chapter, Chapter 4 *Simulation Tool Design*, we further discuss in detail the implementation and design of the software tool used in the methodology.

## 3.1 Research Question Revisited

In Chapter 1 *Introduction*, we discussed central research question which arose from a gap identified in the literature concerning communication agents and their performance. Agents that work together do better in terms of their performance in

attaining group goals when they communicate. However, there is a lack of detailed analysis in the literature on the cause-effect relationship between communication and performance. The central question in this research is then, *exactly where and how the resultant effect is realized within the simulation of the agents themselves*.

In Chapter 1 *Introduction*, we more formally framed the central question as:

If we denote the set of agents in a Multi-Agent System (MAS) by *X*, then a MAS with *n* agents is given by

$$X = \{x_1, x_2, \ldots x_n\}$$

Assuming communication is a function of discrete time, we denote a message between two agents $x_i$ and $x_j$ at time *t* as

$$m_{i,j}(t): x_i \rightarrow x_j$$

then the total communication for *X* at time *t* is then given by

$$M(t) = \{m_{i,j}(t)\}$$

and the messages received by $x_j$ at time *t* is denoted by

$$M_j(t) = \{m_{i,j}(t)\}_{i=1\ldots n, i \neq j} \subseteq M(t)$$

Now we know from previous research that allowing agents to communicate generally increases the groups chances of success, thus the communication *M(t)* at time *t* must have a discernible effect on the group at time *t+1*, say $\Delta_{t+1} X$ . As the simulation progresses in time, we can denote the change in the group from time *t* to time *t+1* as

$$\Delta_{t+1} X: \; X_t \overset{M(t)}{\rightarrow} X_{t+1}$$

To analyze of the effects of agent communication as time progresses, in particular, the effects on the group *X*, we consider the progression of changes $\{\Delta_t X\}_{t=1\ldots T}$ where the end of the simulation time is denoted by *T*. These changes will be realized in the individual agents, thus $\forall x_i \in X$, $\Delta_{t+1} X = \{\delta_{t+1} x_i\}$ where $\delta_{t+1} x_i$ are the resultant changes in $x_i$ at time *t+1* caused by $M_i(t)$. Thus the central research question then becomes a complete description (or analysis) of

$$\{\Delta_t X\}_{t=1\ldots T}$$

## 3.2 The Research Process

There are many incorrect usages of the term '*research*', and there are many things that research *is not*, as the term is often abused through lack of understanding. We can regard formal research as process we undertake to systematically resolve answers to questions or problems, with the support of data we collect through this process (Leedy, 1997). Leedy (Leedy, 1997), outlines eight distinct characteristics of formal research:

1. Research originates with a question or problem
2. Research requires a clear articulation of a goal
3. Research follows follows a specific plan or procedure
4. Research usually divides the principal problem into more manageable sub-problems
5. Research is guided by the specific problem
6. Research accepts certain critical assumptions
7. Research requires the collection and interpretation of data
8. Research by its nature is cyclical (or more precisely helical)

**Research originates with a question or problem**: The world is fill of questions, and any formal research begins with a question that developed from something unexplained, perhaps from unexplained phenomenon, or observation of some process,

or perhaps follow-on questions from past research. By posing such questions we set goals, and thus set in motion the impetus for research to take place. It serves as the catalyst for the research process.

**Research requires a clear articulation of a goal**: In order to proceed in the research process, we must be able to articulate the question in a formal way. This is a crucial step, since only by an unambiguous statement of the question can we proceed forward in a formal structured manner. If the question remains ambiguous, then we cannot be sure we are approaching our problem from the correct perspective. Additionally, even if we do continue and arrive at conclusions or answers, we cannot be certain that these will unambiguous in themselves.

**Research follows follows a specific plan or procedure**: like many other endeavors, research is an activity that must be planned. Proceeding with the research process without a defined and carefully chosen plan could lead to fruitless efforts, or worse, results that are incorrect or ambiguous. By planning the research we can carefully control the parameters of the research that will lead to the generation of data.

**Research usually divided the principal problem into more manageable sub-problems**: with most research problems, it will be difficult at best for the researcher to deal with the core research question as a whole. Most research questions are tackled piecemeal, as are many problems in all areas of human endeavor. The principal behind this approach is that the *whole is composed of the sum of the parts*. The research question itself should suggest an approach to dividing the research into appropriate sub-problems as it is being articulated.

**Research is guided by the specific problem**: once the research question is decomposed into a number of sub-problems, and each of these in turn is articulated, we can then make a reasonable hypothesis about the outcome, or relationships of the outcome to the data. When we view the components through the hypothesis, it may guide the enquiry to possible sources of data that can aid in resolving these sub-questions, and hence the original research question. Additionally, the nature of the question will guide the approach or methodology (discussed in the next section) which

becomes the framework under which the research is conducted.

**Research accepts certain critical assumptions**: in many cases it is valid to make certain assumptions within a research project. Assumptions are conditions within the research that we take for granted, or perhaps there are too many variables within the research so we fix some of them based on sound reasoning. However, we must be careful in any assumptions made, since if the assumptions are not valid then the research should not proceed. We cannot simply state the assumptions we make in research, they must be justified (Simon, 2011). Carver, Van Voorhis and Basaili (Carver et al., 2004) additionally discuss the impact of assumptions on experimental validity.

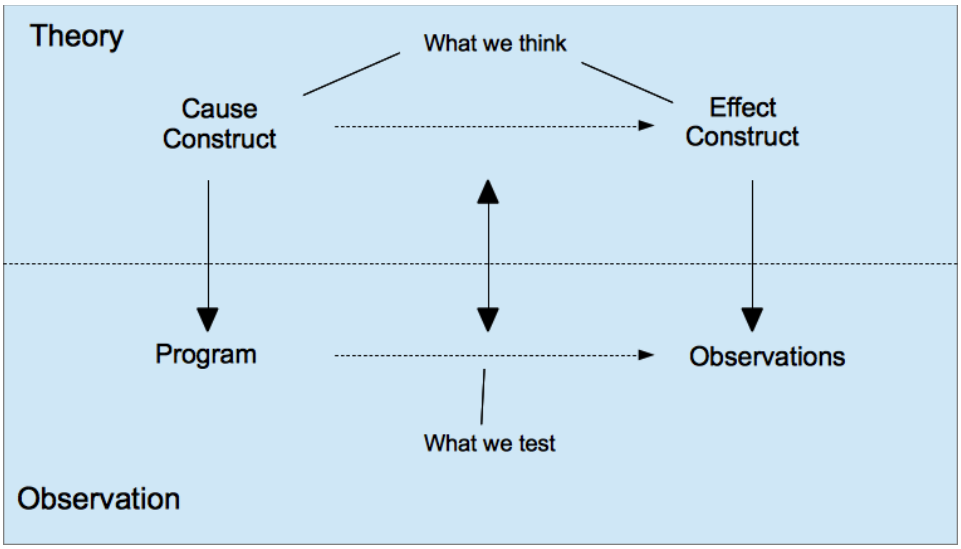


*Figure 3.1: Theoretical constructs used in research (adapted from (Carver et al., 2004))*

Figure 3.1 which is adapted from (Carver et al., 2004) shows the theoretical constructs used in research. Invalid assumptions can cause potential problems between the Cause construct (from a cause-effect relationship) and the Program construct (the tool that represents the realization of the cause construct). Furthermore, invalid assumptions can also effect observations realized from the application of the program or tool. These can become a source of internal validity problems for the research.

**Research requires the collection and interpretation of data**: in any research project we collect data which will then be interpreted, to either support or refute any hypothesis made. This will ultimately be used in answering the original research question. In fact, according to Leedy and Ormrod (Leedy and Ormrod, 2001), we can only undertake research as an approach to answering a problem when there is data to support it (Hardy, 2005).

**Research by its nature is cyclical**: the answers to research questions can be very elusive, and even when it seem they are not, they can more often than not be inconclusive. Thus the research process often given rise to more questions which need to be answered thus creating a cyclic process, as shown in FFigure 3.2.

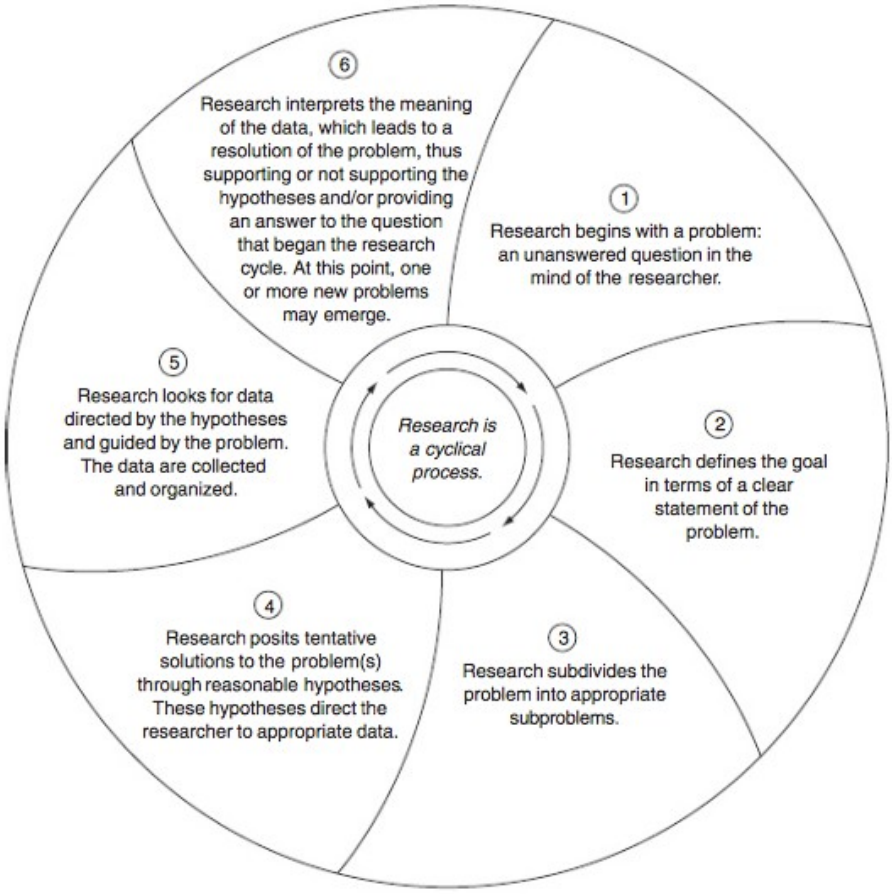


*Figure 3.2: The research cycle (Leedy, 1997)*

The two-dimensional circular representation of the research process from Leedy (Leedy, 1997), shown in FFigure 3.2 can be improved upon to highlight the continuing

nature and accumulation of knowledge through the research process. Walliman (Walliman, 2005) developed a spiral diagram of the research process based on Leedy's circular diagram shown in FFigure 3.3. This diagram still strongly illustrates the cyclical nature of the research process, but more importantly also emphasizes the accumulation of knowledge through this process. As we turn through each twist in the spiral, the knowledge we accumulate is raised, and this in turn inspires further research questions and lines of enquiry (Walliman, 2005).

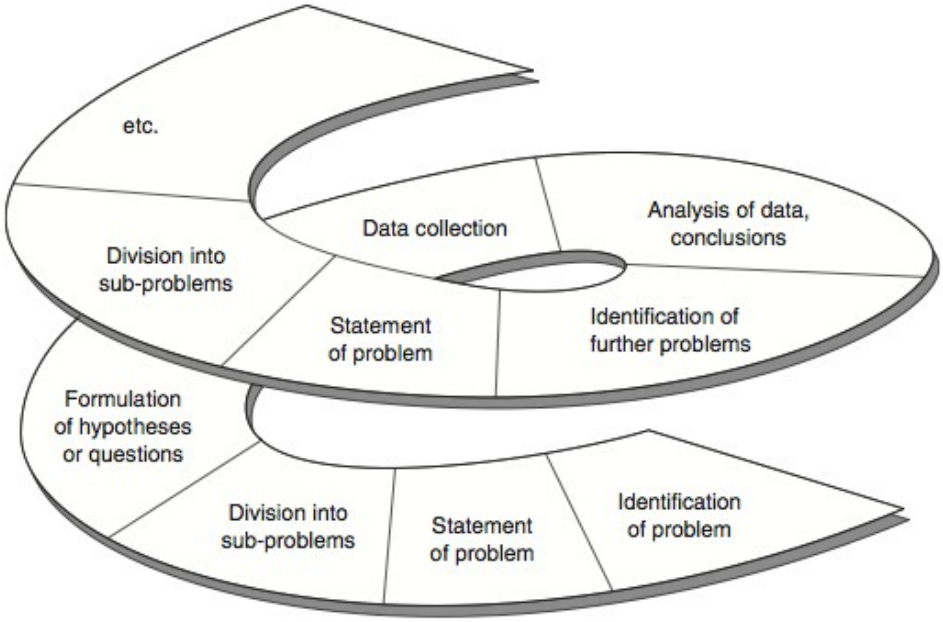


*Figure 3.3: Spiram diagram of the research process (Walliman, 2005)*

Walliman additionally provides a diagram showing the simple relationships between the five main elements of the research process. These are the *Situation,* the *Research Topic*, the *Research Methods*, the *Data* and the *Conclusions.* The *Situation* represents the state or the relative space in which the researcher is observing or studying a particular phenomenon.The *Research Topic* is of course the general area under which the research questions are framed. The *Research Methods* represent the general approach taken in the research process, and these are discussed in detail in the following section. The *Data* is what we collect while investigating the research problem, and the *Conclusions* are the generalizations and results of analysis and interpretation of the Data, This general relationship is shown in Figure 3.4.

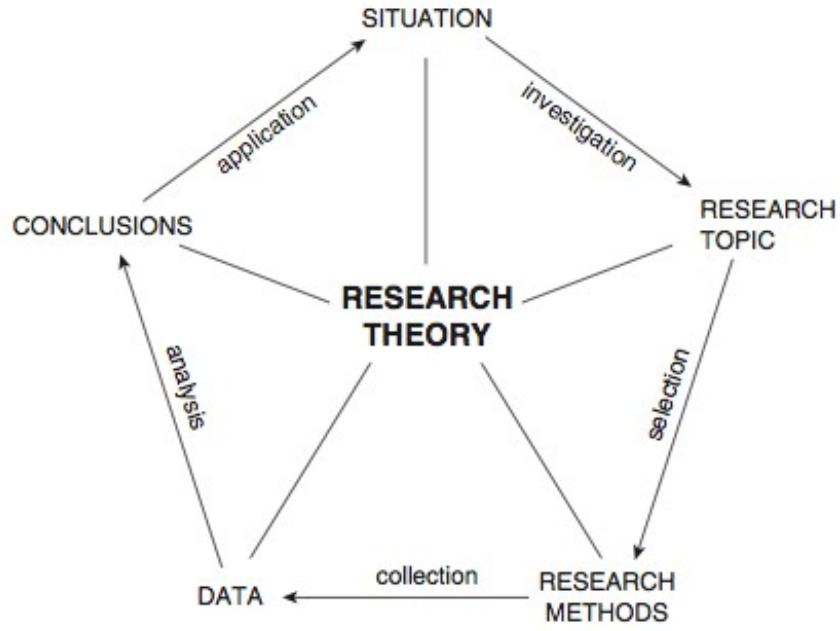


*Figure 3.4: Relationship between elements of the research process (Walliman, 2005)*

In essence, all formal research is modeled on the *Scientific Method (Leedy, 1997, Shavelson and Towne, 2002)*. The beginnings of scientific research can be found in early recorded works from ancient Greece, Persian, Indian, Chinese and European cultures (Carpi and Egger, 2008). However, the basis of what we term the *Scientific Method* came from the scientific revolution in the renaissance period during the 16th - 17th centuries. The beginning of the scientific revolution is often cited as having begun with the publication of '*De Revolutionibus Orbium Coelestium (On the Revolutions of the Heavenly Spheres)*', by Nicolaus Copernicus (Carpi and Egger, 2008). This method was based on the rigorous collection of data from observed phenomenon along with a way of thinking known as inductive reasoning.

Although formal research may be based broadly on the scientific method, the classic view of the scientific method as :

Observation → Question → Hypothesis → Experiment → Conclusion

is a little misleading. The *experiment* component in the classic view of scientific method is not always applicable, and this is where research projects will differ, in the *methodology*, data collection and analysis

## 3.3 Research Methodology

The Research Methodology is the general framework which guides us in systematically solving the research problem (Kothari, 2004). Every research project is based on such a conceptual framework (Walliman, 2005, Bickman and Rog, 2008), and the methodology provides the general approach we take to the collection and analysis of the data. According to Leedy, (Leedy, 1997), the scientific method to solving research problems is only viable when there is data available to support it. The data and the methodology are then tightly bound together in an interdependent relationship, thus the research methodology we adopt will depend on the nature of the data we will collect during the research process.

### 3.3.1 Qualitative vs Quantitative

It has been said that there are almost as many research methodologies as there are research projects, as each is different in nature and may require different data collection methods as well as different analysis techniques, but we can generally classify most research methodologies into two broad categories, *Qualitative* and *Quantitative* approaches (Dawson, 2005, Leedy, 1997, Carpi and Egger, 2008, Kothari, 2004). Quantitative research methods are primarily concerned with the collection of numeric type data and the analysis of such data so things can be measured numerically. Quantitative methods had their roots in the natural sciences where scientists are concerned with how something is constructed or works. Qualitative methods methods had their origins in the social sciences where scientists are concerned more with the increase of our understanding in these areas and not with explanations (Berndtsson et al., 2002). The types of data collect in qualitative research projects is not as quantifiable as that in quantitative research and can be in the form of images or textual pieces such as interviews and case studies.

Quantitative methods are more often found in the physical sciences, while qualitative methods are more often found in research projects associated with the social sciences, though this division is not fixed. Table 3.1 Presents the major distinguishing features between quantitative and qualitative research methods.

*Table 3.1: Distinguishing Characteristics of Quantitative and Qualitative methodologies (Leedy and Ormrod, 2001)*

| Question | Quantitative | Qualitative |
|---|---|---|
| What is the purpose of the research | • To explain and predict<br>• To confirm an validate<br>• To test theory | • To describe an explain<br>• To explore and interpret<br>• To build theory |
| What is the nature of the research | • Focused<br>• Known variables<br>• Established guidelines<br>• Pre-determined methods<br>• Somewhat context-free<br>• Detached view | • Holistic<br>• Unknown variables<br>• Flexible guidelines<br>• Emergent methods<br>• Context-bound<br>• Personal View |
| What are the data like and how are they collected | • Numeric data<br>• Representative large sample<br>• Standardized instruments | • Textual and/ or image based data<br>• Informative small sample<br>• Loosely structured or non-standard observations and interviews |
| How are the data analyzed to determine their meaning | • Statistical analysis<br>• Stress on objectivity<br>• Deductive reasoning | • Search for Themes and categories<br>• Acknowledgment that analysis is subjective and potentially biased<br>• Inductive reasoning |
| How are the findings communicated | • Numbers<br>• Statistics, aggregated data<br>• Formal voice, scientific style | • Words<br>• Narratives, individual quotes<br>• Personal Voice, literary style |

Four of the most common research methods utilized are that of action research, experimental research, case study and survey methods (Dawson, 2005), though Leedy (Leedy, 1997) provides a more exhaustive list. It is beyond this chapter to provide an exhaustive list of research methodologies however the four common methodologies will be briefly discussed.

#### *3.3.1.1 Action Research*

Action research involves the carefully documented and monitored attempt by the researcher to solve a local problem. This is a form of applied research where the researchers actions are meant to change a local situation in order to gain a better understanding of *practice* and methods for improvement (Dawson, 2005, Leedy, 1997). There is a danger that the researcher may become too close to the problem and become too consumed with achieving the results and disregard the essential impetus of the research project.

Action research is a participatory research method where the researcher is not merely an observer, but an active participant. Figure 3.5 Shows the cyclic nature of action research.

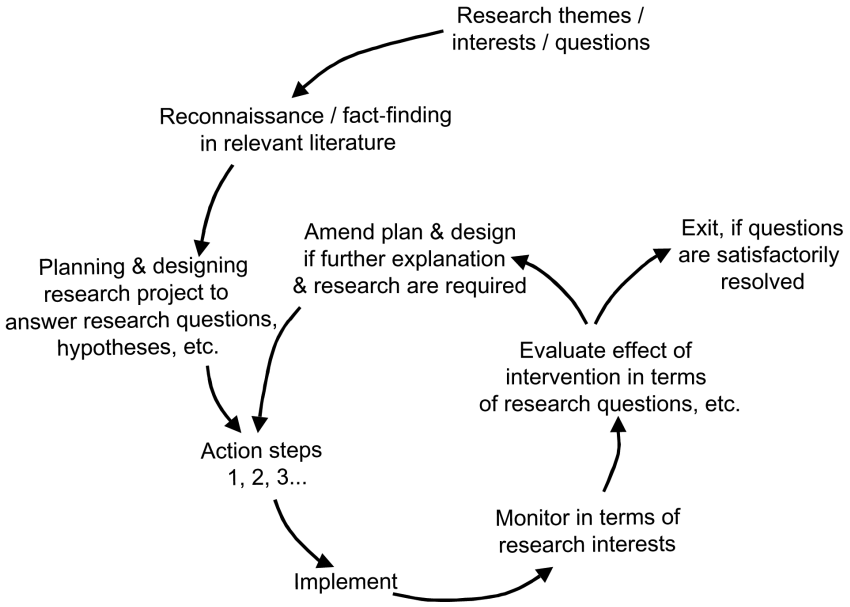


*Figure 3.5: Cyclic nature of action research (McKay and Marshall, 2001)*

#### *3.3.1.2 Experimental Research*

Experimental research is normally undertaken in artificial environments or laboratories where the researcher investigates cause-effect relationships with the

ultimate aim of being able to predict phenomenon given the values of certain state variables investigated during the research. One of the distinguishing features of experimental research is the researcher does not simply observe and record the variables, but manipulates them in a systematic way to observe the effects (Luzzi, 2005b). This aids in the understanding of the cause-effect relationship.

This methodology normally requires a control group against which the performance of a test group is measured. In the test group, one or more independent variables are manipulated and the effects on the test group are then observed and measured against the control group, subject to the same external conditions. The effect of manipulating the independent variables can then be quantized and later used for prediction.

A second distinguishing feature of experimental research is the control of extraneous variables. The extraneous variables in a specific experimental research project are all those that are deemed to not be of interest to the researcher. These are usually many in number and they can be managed by either trying to keep them constant between the control and test groups, or even by randomizing them as much as possible. Experimental research methods tend to be higher in internal validity (Luzzi, 2005b), but are subject to a number of limitations.

Sometimes it is difficult to undertake experimental research because the manipulation of the independent variables cannot be performed in live situations for practical reasons (as in this research, manipulating communication of teams during a combat scenario). Additionally, in trying to control the extraneous variables the researcher is sometimes forced to contrive artificial scenarios which may limit the applicability of any determined results.

#### *3.3.1.3 Case Study Methods*

The case study is one of the oldest research methods and data is gathered through an in-depth investigation into a particular person, a group of people or an organization. It captures the complexity of a single case (Johansson, 2003). The case study is a

flexible research method and is usually conducted in order to try and reveal "Universal Truths" by studying a particular scenario in-depth (Luzzi, 2005a).

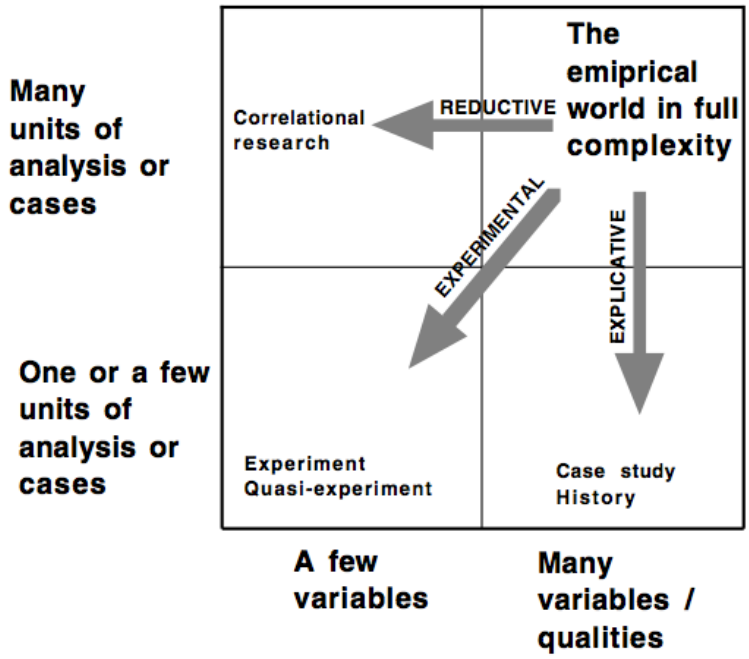


*Figure 3.6: Figure showing the case study in a framework methods used to reduce data complexity (Johansson, 2003)*

Figure 3.6 shows the case study method in a framework of methodologies which focus on different strategies to reduce data complexity when undertaking an empirical investigation (Johansson, 2003). The case study is an explicative, or interpretive methodology which seeks to reduce complexity by focusing on one case but envelopes many variables to account for the context. The case study methodology developed from the social sciences, but has become a methodology utilized in many disciplines and applied to many types of research problems. The purpose of the case study to explain the 'why' (Luzzi, 2005a).

The case study study method has some problems which may limit its usefulness in specific situations. There is always the danger of observer bias, where the researcher background and beliefs lead them to misinterpret their observations and report incorrectly. The researchers should be trained to be objective observers. Additionally, the case study represents a very narrow focus on an individual, group or specific

organization. While this may be necessary to reduce complexity, in may instances, enough data will not have been collected in order to generalize findings to a wider population.

#### *3.3.1.4 Survey Methods*

Survey methods have been utilized since the 19th century, and are used in the research process within many disciplines to gather data by asking questions. A survey is '*A systematic method for gathering information from (a sample of) individuals for the purposes of describing the attributes of the larger population of which the individuals are members*' (Enanoria, 2013). Information is gathered by presenting individuals with a structured set of questions, or closely related, by having an interviewer as a series of questions and recording the answers.

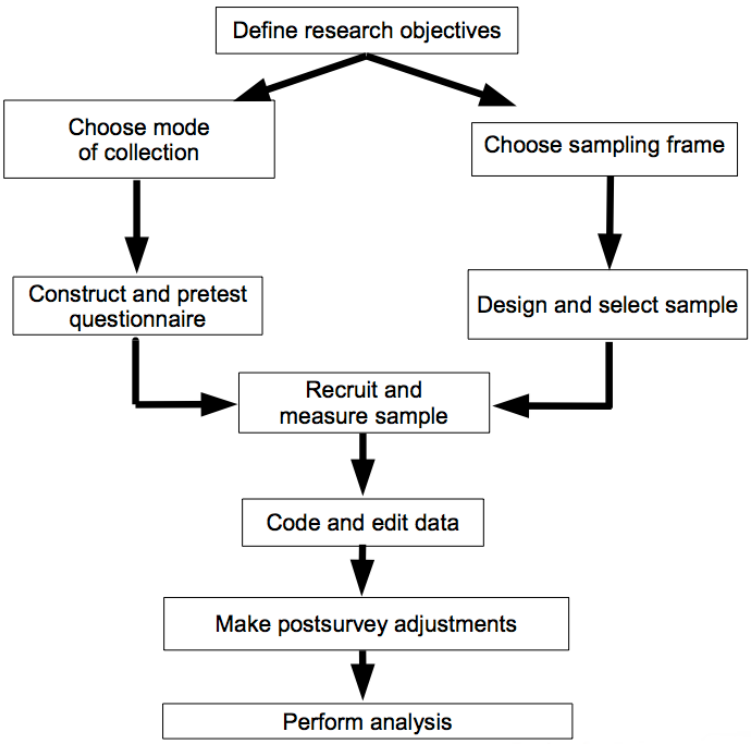


*Figure 3.7: The survey process (Groves et al., 2004)*

Once the data is collected it can be analyzed using statistical techniques to describe various characteristics of the sample such as frequency, distribution and

averages (Leedy, 1997). By extrapolation, the researcher can then infer similar characteristics of the larger population. Figure 3.7 presents a diagram showing the overall survey process. As with other methodologies, there are a number of advantages and disadvantages offered by the use of survey methodologies, as well as challenges in constructing the survey instrument.

In comparison to other methods, data is usually reliable and variability can be reduced by the number of surveys conducted. Additionally, surveys can yield a broader range of data to be analyzed. However, there are also a number of limitations including effects introduced by the response order, bias built into questionnaires, human bias in answering questionnaires, unwillingness to provide answers, and inability to provide answers including any semantic difficulties with the survey questions. The construction of the survey questions are of critical importance and should be considered carefully.

### 3.3.2 Simulation as a research Tool

As discussed in *Chapter 1 Introduction*, this research is concerned with studying the effects of communication on a group of agents in a battlefield scenario. The central research question has been restated in *3.1 Research Question Revisited*, and is a complete description of :

$$\{\Delta_t X\}_{t=1\ldots T}$$

which is the change in the group of agents $X$ as time progresses from $t = 1... T$ in a battlefield environment due to the effects of $M(t)_{t=1\ldots T}$ which is the total communication within the group $X$ at time $t$.

From our brief discussions on research methodologies in the previous sections, a quantitative research methodology would be best suited for this problem. In particular, an experimental approach where we can have a control group using no communication, and a test group which uses communication would be desirable as we could measure the discernible difference between the outcomes of control group and the test group. However, there are specific types of problems where it is not possible to use the more

traditional research methodologies, either due to practical reasons or even ethical considerations. In the research problem outlined here, it is not practical to utilize such techniques in a combat scenario.

Agent-based modeling is a relatively new analytical method for modeling in the social sciences (Gilbert, 2007). While building a model and testing it is not a new concept, agent based modeling allows us to build models which enable individuals to be directly represented as well as the interaction between them. This is a considerable improvement over equation based modeling and we can now model complex adaptive systems using such simulations and search for emergent behavior. However it has been suggested by some researchers that simulations offer little in the way of theory development and are just models of real systems that just replicate observed phenomenon and are so abstract that they can give limited insights at best (Davis et al., 2007, Chattoe, 1998, Fine and Elsbach, 2000).

In contrast, simulation as a methodological approach has been referred to a '*third way of doing science*' (Axelrod, 2003, Gilbert, 1996, Ramanath and Gilbert, 2003). Simulation can be contrasted with two standard methods of reasoning: *induction,* the search or patterns in empirical data; and *deduction,* which involves specifying a set of assumptions and then deriving consequences from the assumptions. However, simulation begins with a set of assumptions (coded into the simulation model) but does not derive or prove anything, rather the model generates large amounts of data that can then be analyzed inductively (Axelrod, 2003). Another difference is that with induction, data is normally gathered from surveys, interviews and case studies, but the data generated by a simulation is derived from a set of clearly defined rules built into the simulation.

Axelrod (Axelrod, 2003), discusses the value of simulation according to the purposes to which it may be put:

- Prediction – simulation can utilize complex inputs, execute the simulation based on programmed rules and generate the outputs as predictions

- Performance – simulations can be used to perform and study many different types of scenarios such as medical diagnosis, speech recognition, optimization of traffic flow and the inclusion of artificial intelligence techniques will help the simulation mimic to some extent the way humans deal with these tasks.

- Training – some of the first uses of simulation were in training, and modern simulation techniques extend this to more sophisticated training techniques to include flight simulators, military combat missions, stock trading, and many others.

- Entertainment – as an extension of training modern computer games are a source of entertainment and in some cases, based on actual training simulations.

- Education – many training simulations can also be used in education, such as stock market trading, SimCity (which teaches younger students about development and planning), clinical simulations for medical students, and social simulations to teach about human phenomenon and reactions.

- Proof – in some cases simulations aid in providing an existence of a proof, or of a proof of concept. Conway's game of life was used to prove that complex behavior could be demonstrated by the application within a simulation of a very small set of of simple rules. Boids (Reynolds, 1987), was used to prove that complex flocking behavior in birds could again be demonstrated by a small set of simple rules.

- Discovery – discovery, prediction and proof are the key elements when talking about the value of simulation as a scientific methodology. Prediction can help validate and improve a model, but discovery has been the cornerstone of the use of simulations in social sciences for some time. Simulations in such areas can help discover new relationships from very simple models and rules. (Axelrod, 2003)

Gilbert and Troitzsch (Gilbert and Troitzsch, 2005) proposed a simple architectural framework for the simulation methodology, shown in Figure 3.8, to explain the logic of the methodology. In Figure 3.8, beginning with the *Target*, representing the real world system, the researcher abstracts the characteristics of the real world system they wish to study and uses these to build the model. Simulation runs of the model then produce the simulated data which can then be compared and correlated against data collected from observing the real-world system.

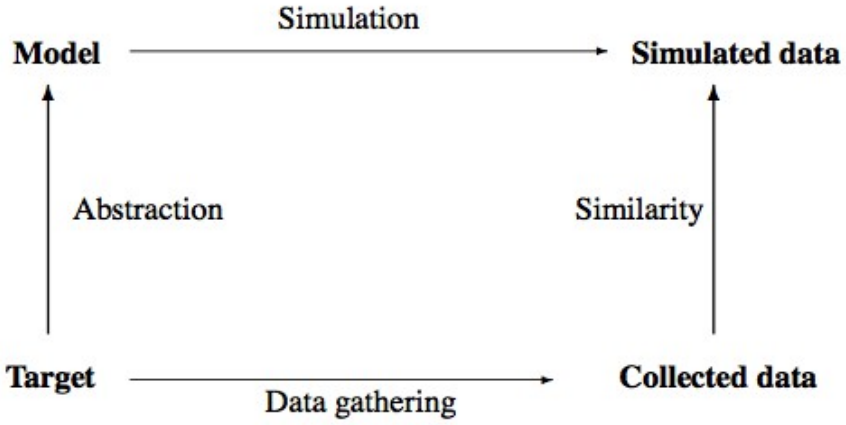


*Figure 3.8: Framework for simulation methodology (Gilbert and Troitzsch, 2005)*

Gilbert and Troitzsch (Gilbert and Troitzsch, 2005) point out that the logic underlying the use of simulation models in their framework in Figure 3.8 is much the same as the logic underlying statistical modeling as a research method. In statistical modeling, a model is constructed as an abstraction from a social process, parameter estimation is used to derive predicted data which in turn is compared for similarity to collected data from the original social processes. Both statistical models and simulation models can be used for prediction and illustration (Gilbert and Troitzsch, 2005).

Ihrig (Ihrig, 2012) argues that the framework presented by Gilbert and Troitzsch in Figure 3.8 illustrates succinctly the logic behind simulation as a research method, but it doesn't capture completely the simulation research process experienced by

researchers. Ihrig derived an expanded architectural framework for simulation methodology and this is shown in Figure 3.9. This expanded framework extends the work of Gilbert and Troitzsch (Gilbert and Troitzsch, 2005) to provide a more complex process model that describes the relationships between all the building blocks of models which are often used to simulate complex systems (Ihrig, 2012).

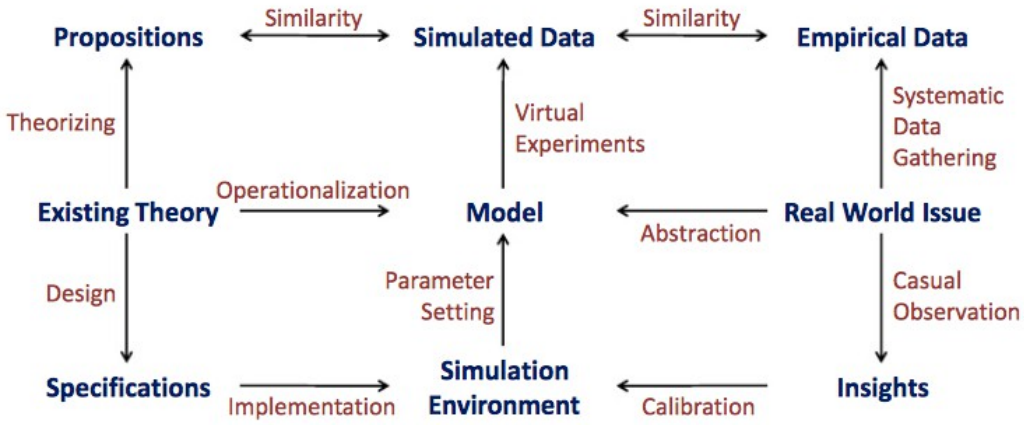


*Figure 3.9: Expanded framework for simulation methodology (Ihrig, 2012)*

In the expanded framework in Figure 3.9, the process of building the model begins with an existing theory of the real-world system and the simulation environment is built from specifications derived from the theory and calibrated by insights gained from observations of the real-world system. This depicts the real-world grounding of the model. The model is derived from the simulation environment via determining the parameter settings. The simulated data collected from the simulation of the model can then be contrasted to empirical data collected from the real-world system and against propositions derived from the underlying theory. When calibrated, the simulation can then be used for confirmation of theory and for prediction.

What distinguishes this expanded framework is that it provides a stronger theoretical foundation for the simulation research when models are based on an existing theory and calibrated using insights. Of course, if an existing theory is lacking, it is possible that the simulation can be used to help develop theory, but this is generally not the case.

Ramanath and Gilbert (Ramanath and Gilbert, 2003), have developed a generic model to highlight stages undertaken in a simulation based research projects. These are generic in the sense that other researchers have identified similar steps. One of the perceived weaknesses of the simulation method is often the poor statement of the initial model foundations and conceptualization. However if we undertake the steps of model conceptualization and design within the framework provided in Figure 3.9 we can reduce or eliminate criticism in this area.

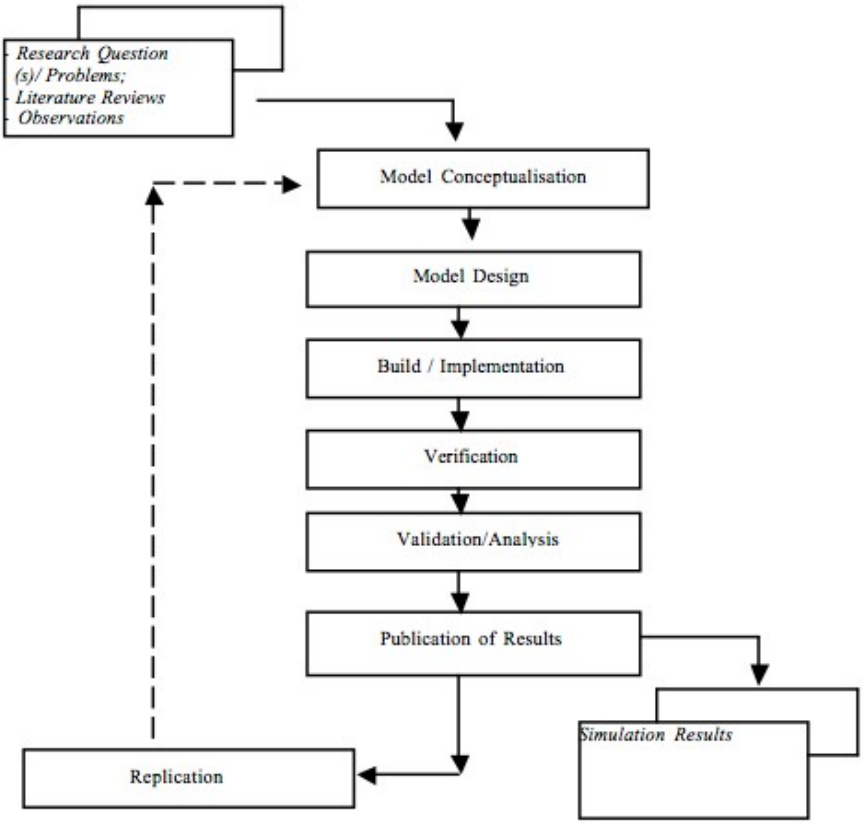


*Figure 3.10: Generic stages in the simulation research process (Ramanath and Gilbert, 2003)*

Verification and validation are also important steps. Given that simulation models are abstractions for the real-world, they often have poor external validity. By utilizing the expanded framework in Figure 3.9 when developing the model we can create strong internal validity, and this in turn addresses a common weakness in empirical research of poor theoretical basis (Davis et al., 2007).

Simulation is a form of computational modeling (Gilbert, 2007), which highlights one of the greatest strengths of simulation as a research tool, that of experimentation. Researchers can systematically experiment in a controlled environment (Davis et al., 2007), and move the simulation forward or backward in time. This is particularly useful when seeking to study longitudinal phenomena. Additionally, simulation allows us to isolate the human system and any ethical problems of experimentation (Gilbert, 2007). The experiments can be established and repeated as many times as desired without the ethical considerations associated with the underlying real-world system.

## 3.4 Research Design

The goal of this thesis is to develop and test a model for cooperative agent communication utilizing a reinforcement learning algorithm in a multi-agent environment . This then will provide us with the means to examine in detail the cause effect relationship between agent communication and group outcomes, filling a gap in the literature. In this section we provide details of the research design and the methodology chosen as the approach used to collect the data required for analysis.

The methodology used to undertake this research is that of simulation. As discussed in the previous section, this research is undertaken to study the effects of communication on the success rate of a group of agents undertaking a task situated in a hostile environment. This research began at the Australian Defence Force Academy located in the A.C.T. (Australian Capital Territory). The environment is that of a battlefield where two groups of opposing agents try to eliminate each other. In this scenario an experimental approach would suit the problem to be undertaken. However it is impractical to approach the problem by devising experiments directly on the underlying domain. By utilizing a simulation methodology we can abstract out the important features of the target domain to construct a model where these important features and be modeled and experimented with so data can be collected and analyzed. The remainder of this section will discuss the design aspects of the research experiments.

An experimental approach was undertaken within the simulation model. In order to study the effects of communication of a group of agents *X*, two groups of agents were constructed, a *control group* and a t*est group* (*X*). The *control group* represented a group of agents that were pre-programmed with specific behavior (or goals) and was the equivalent of a highly trained group of combatants. The test group were not trained but had access to a set of possible actions and were governed in their behavior by an artificial intelligence learning algorithm called reinforcement learning. Utilizing this algorithm the test group would experiment with actions and receive positive or negative rewards from the simulation depending on whether the actions would lead to a success or otherwise. The major elements of the simulation are shown in Figure 3.11.

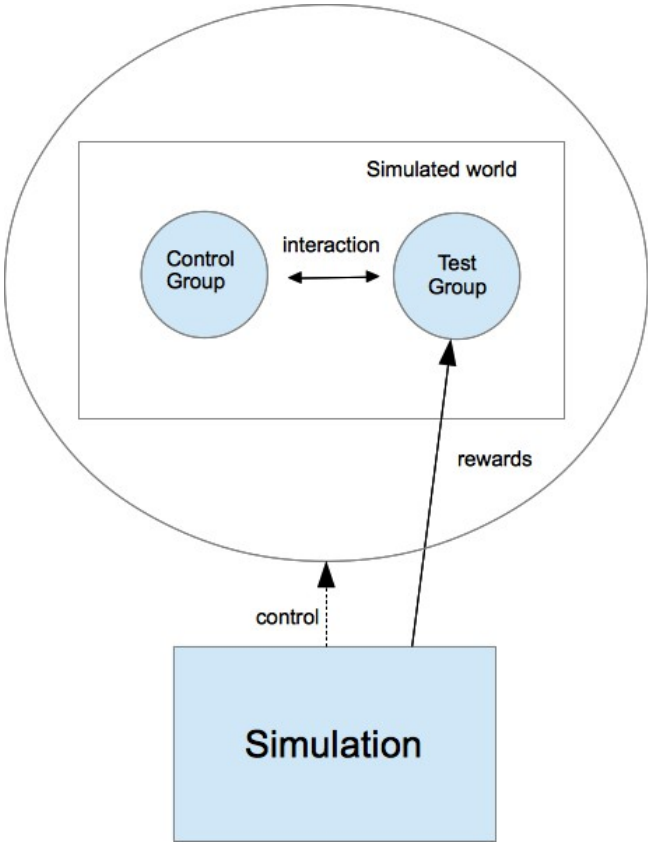


*Figure 3.11: Major elements of the simulation*

Each group of agents had their own goals and would interact with each other through the predefined actions of the simulation. The goal of each group was to eliminate the other group in a fight for survival. The control group did not communicate

with each other, but each agent in the control group was governed by the same behavior, thus there was a form of implicit cooperation within the group In the test group, the communication was treated as the independent variable and experiments were devised to vary the communication so its effects on the success of the group could be measured. The goals of the test group were not built into the agents as in the control group, but were governed by the reward function in the simulation which rewarded the agent positively if they caused damage to an agent from the control group, or negatively if their action led to damage being inflicted on them from an agent in the control group.

Four basic experiments were devised and data was collected on each experiment via simulation runs. A simulation run was the execution of the simulation model for a specific period of time using the parameters of the specific experiment. The four experiments devised were labeled as

- Benchmark Trial – in this experimental trial the test group were not allowed to communicate. Rather, each agent acted independently governed by the reinforcement learning algorithm, receiving positive or negative rewards from the simulation depending on the results of their actions. The purpose of this experiment was to benchmark the level of success of the test group without communication against the control group.

- Comms 1 – in this experiment, the test group were allowed to communicate with each member of the same group by sharing their knowledge of the actions they performed and the rewards they received from the simulation for those actions. However, in this trial the test group agents could only communicate the positive rewards and the actions taken to get a positive reward.

- Comms 2 - in this experiment, the test group were allowed to communicate with each member of the same group by sharing their knowledge of the actions they performed and the rewards they received from the simulation for those actions. In this trial the test group agents could only communicate the negative rewards and the actions taken to get a negative reward.

- Comms 3 - in this experiment, the test group were allowed to communicate with each member of the same group by sharing all the rewards they received from the simulation and the actions that they undertook in order to receive the reward. The *comms1* and *comms2* experiments were designed to see if positive or negative rewards only were sufficient to be successful against the control group.

For each experimental run, the simulation recorded each movement, action and reward that each agent undertook in each time step. The simulation was designed as a discrete time simulation, and at each time-step each agent was allowed to observe the world and decide on a course of action. The actions were then applied randomly and then each agent received their reward or otherwise depending on the result of the action.

There was a question of how many simulation runs was needed for each of the trials, and the number will be dictated in large by the nature of the simulation. The question of whether 100, 1000 or 10000 runs be sufficient to gather enough data for each run in order to analyze the results with confidence depends on the nature of the simulation. This simulation was somewhat different as reinforcement learning was utilized in a multi-agent system. Reinforcement learning was developed initially as an artificial intelligence technique to be utilized for a lone agent to act in an environment and receive rewards from the environment. By utilizing reinforcement learning in a multi-agent system the actions of the different agents would influence the state of the other agents thus creating a non-markov process. While this is a valid technique for multi-agent systems, it meant that the simulation runs would not converge, and thus for each agent optimizing their reward became a moving target.

It was determined to utilize experimental tests of 2000 runs per experiment. Thus for each of the four experiments above, *benchmark*, *comms1*, *comms2*, and *comms3* 2000 simulation runs were conducted and the results stored for later analysis. As the simulations did not converge, there was not much to be gained from utilizing more that 2000 runs as the data generated was extremely large and each run varied from each other run without convergence. Two thousand runs was enough to analyze and in some cases average results so a clear picture of the simulation trends emerged.

Each of the experimental trials was then analyzed and comparisons were made. These were then used to develop *Knowledge Landscape* diagrams so a three dimensional image of the learned behavior of agents in the experimental trials could be viewed in order to reason about the success of the communication within the test group.

The implementation details of the simulation tool constructed are provided in *Chapter 4 Simulation Tool Design*. The initial simulation utilized a simulation tool called Crocodile developed at the Australian Defence Force Academy (Barlow and Easton, 2002), however the tool was redesigned and re-written for this research to allow for the incorporation of agents utilizing Artificial Intelligence techniques and to allow for the inter-group communication between agents.

# 4

# SIMULATION TOOL DESIGN

*A theory is something nobody believes, except the person who made it. An experiment is something everybody believes, except the person who made it*

– *Albert Einstein*

Agent-based modeling and simulation while relatively new, has become a popular simulation tool used for the analysis and simulation of systems from many diverse problem areas. As discussed in previous chapters, agent-based simulations are replacing many forms of equation-based modeling, as these types of simulations cannot take into account the adaptability of the elements in the model. Additionally, using these types of simulations we can now model complex systems by implementing directly the elements of the model as agents and defining the interactions between them. It is the non-linear interactions between the elements of a system that essentially makes it complex, making it difficult to model these systems using the traditional equation-based modeling techniques.

In this Chapter we detail the design and construction of the simulation tool which is used in this research. The simulation tool is designed to study the effects of communication on a group of agents in a complex system who are trying to achieve a common goal. This research began at the Australian Defence Force Academy (ADFA) and thus was grounded with a military theme. The agent-based system in question represents an abstract land combat scenario involving two opposing forces who battle to eliminate each other. One of the opposing forces allows the members to communicate

their successes or otherwise to each other and it is the effects of this communication on the success of the group that forms the focus of this research. While the underlying domain of the simulation is a military based problem, the application and the simulation itself are general in nature and can be applied to a multitude of multi-agent scenarios.

## 4.1 Language and Versioning Details

The multi-agent simulation software was built using the Java[1] programming language, Java SE 6. However most of the code is also compatible with Java SE 7, except for some of the GUI Swing based components and some elements of the Java Reflection API which have been deprecated in Java SE 7. The Integrated Development Environment utilized when programming, was NetBeans[2]. NetBeans was developed as an Open Source tool by Sun Systems and later acquired by Oracle. NetBeans is offered under the GNU General Public License (GPL) v2. NetBeans Version 6.9.1 is the version currently utilized in this software tool, though the simulation system will execute on earlier versions and additionally version 7.0.x. The simulation will not at this stage compile without errors when using NetBeans 7.1 or later due to the removal of support for the Swing Application Framework (JSR 296) in version 7.1

NetBeans was chosen as an IDE as it was initially developed as an IDE primarily for the Java language, though it also supports other languages. The NetBeans IDE is additionally written in Java and thus offers the portability of the Java language. This will allow any follow-on research and simulation projects arising as a result of the development of this tool to be available on as many platforms as possible.

The Java language was chosen for a number of reasons, but first and foremost because it is an Object Oriented language designed to run on the Java Virtual Machine (JVM). The Java Virtual Machine has been ported to all major platforms, thus making Java code '*write once, run anywhere*'. It is always important to eliminate any portability issues from research projects, allowing the researchers to concentrate on the fundamental issues of developing the tool rather than become focused on mechanics on implementation. The nature of the underlying problem also indicated that an Object

1 Java is a trademark and product of the Oracle Corporation

2 NetBeans is a product of the Oracle Corporation

Oriented Programming language would be better suited to implement the solution.

The underlying domain of the problem under consideration is a land combat scenario. Land combat exhibits all the elements of a complex system in so much as it consists of many components that interact in a non-linear fashion. Sun Tzu (SunTzu, 2005), an ancient Chinese military strategist recognized this as far back as approximately (722 – 481 BC). Thus equation-based modeling is largely ineffective at predicting the outcome when modeling complex systems due to this non-linear interaction. Small variations in initial conditions can lead to wide variations in the outcome, often termed the '*Butterfly Effect*'. Additionally, when we introduce intelligence into the agents through AI techniques, which more closely models components in the underlying domain, we then introduce the possibility of elements of the system being able to adapt to conditions as the system progresses through time.

Distillations, which represent a class of military simulations that model these types of complex systems use agent-based modeling where the elements of the simulation can be directly represented as agents in the simulation. We can then model the complexity by defining the rules that govern how the agents interact within the simulation. By using an Object Oriented Programming language such as Java, we can easily implement the agents in the simulation as objects, and define the interaction between the agents through message passing between the objects. While we could construct the simulation with a more traditional procedural language, in this case, an Object Oriented language such as Java is more suitable to the problem at hand.

The simulation tool was given the name Ares, named after the the Greek god of war. A description of the simulation tool, it's architecture, the implementation details and functioning is given in the following sections. Sections of the actual source code of the simulation tool will be referenced throughout the remainder of this chapter, and the source listings will be provided in Appendix 1. However, the complete source code has not been provided for brevity.

## 4.2 Architecture of the Simulation

The overall architecture of the simulation was designed following a simple design principle of separating the components of the simulation tool into four broad groupings depending on their role within the simulation. This is depicted in Figure 4.1.

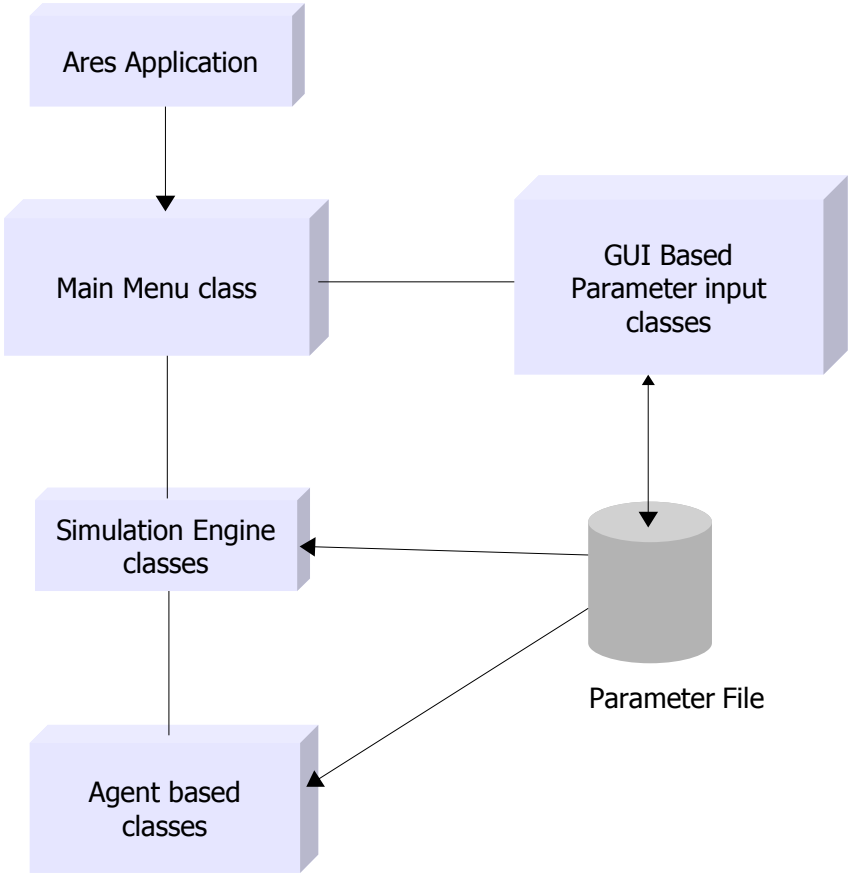


*Figure 4.1: Overall architecture of the Simulation*

The simulation is initiated by the main Ares application class called *AresApp.java*, see Appendix A.1. This is basically a Java loader class, whose sole function is to create an instance of the Ares main menu class and transfer control of the simulation to this class, see *AresView.java,* Appendix A.2. The main menu class uses the Java Swing components to present an event driven Graphical User Interface (GUI) main menu. This is depicted in Figure 4.2. This GUI based main menu then gives access to all elements of the simulation via the use of drop-down menus.

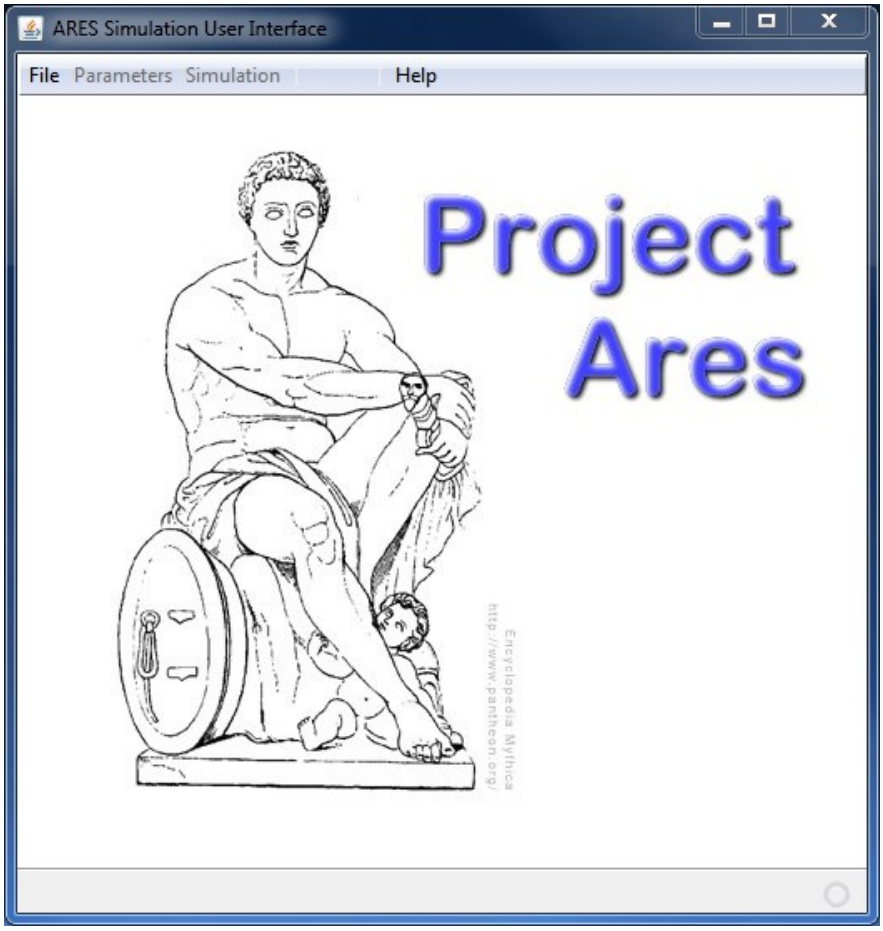


*Figure 4.2: Ares GUI based main menu*

### 4.2.1 Simulation Parameter Control

The simulation is parameter-driven, in so much as the user can specify all the required input parameters to the simulation, save these in a parameter file, and then run the simulation based on the saved parameters. This is indicated in Figure 4.1 as the parameter file plays an important role in feeding into the *Simulation Engine* component all the necessary data required for a '*simulation run*'. We define a *simulation run* as the execution of the simulation for a defined number of time-steps. We will discuss the meaning of a time-step in the simulation in a later section, but it is an abstract notion of time where agents can perform actions. By being able to save the simulation runs in a permanent file, various parameter files can be saved and utilized later if experimental trials need to be repeated.

The main purpose of the GUI-based main menu is to allow a user to create a parameter file, set the parameters using other GUI-based forms, and then to initiate the simulation engine. The '*File*' menu item in the main menu form in Figure 4.2 allows the user to either create a new parameter file, load an existing parameter file, or to exit the simulation. Thus multiple parameter files can be stored and archived for comparative analysis of the results and later utilization. The *Parameters* menu item located on the main menu form allows the user to either set the parameter values of a new parameter file, or to modify the parameter values of an existing parameter file. At the time of this writing, the parameters were divided into three (3) parameter groups which are accessible from the *Parameters* menu. These are:

- *World Parameters*, values on aspects of the world the agents inhabit and operate in throughout the simulation

- *Simulation Parameters*, information on the simulation itself such as time-steps per simulation, number of runs per trial and number of agents per team. The values in this parameter group essentially drive the simulation engine.

- *Agent parameters*, information on the agents themselves, such as their initial location within the simulation. This parameter group also relies on elements of the Simulation Parameter group.

The drop-down menu for the Parameter menu item on the main menu can be seen in Figure 4.3. Additionally, from this menu is the option '*Check Parameter Groups*'. This option must be selected before the '*Simulation*' menu item can be selected. The purpose of this option is to do a final check for the presence of all parameter groups and validate parameter values before the simulation begins. This is a final check against running the simulation with invalid values which could result in erroneous results arising from the simulation. See Appendix A.3 for the *checkPGUI.java* code.


*Figure 4.3: Parameter groups accessible from the parameter menu item*

The structure of the parameter file was encoded to allow for future expansion and the addition of further parameter groups in later research, however as discussed, currently only three parameter groups are encoded. There are many values encoded throughout the simulation for expediency but these can be moved into existing or new parameter groups within the parameter file with relative ease. The structure of the parameter file is detailed in Figure 4.4

### 4.2.2 Parameter Groups

The current data stored for each of the parameter groups is described as follows:

#### *4.2.2.1 World Parameter Group*

The values kept for the *World Parameter Group* minimal and currently are just the size of the world. The world in the simulation is represented by a space with a width and height. Being a distillation, this simulation is an abstract representation of a combat scenario, and no other terrain information is required. This allows us to study the cognitive effects of communication on the agents in isolation to external influences. It is envisaged that in further research, terrain data can be stored for the world, along with other world objects representing obstacles, or cover for the agents.

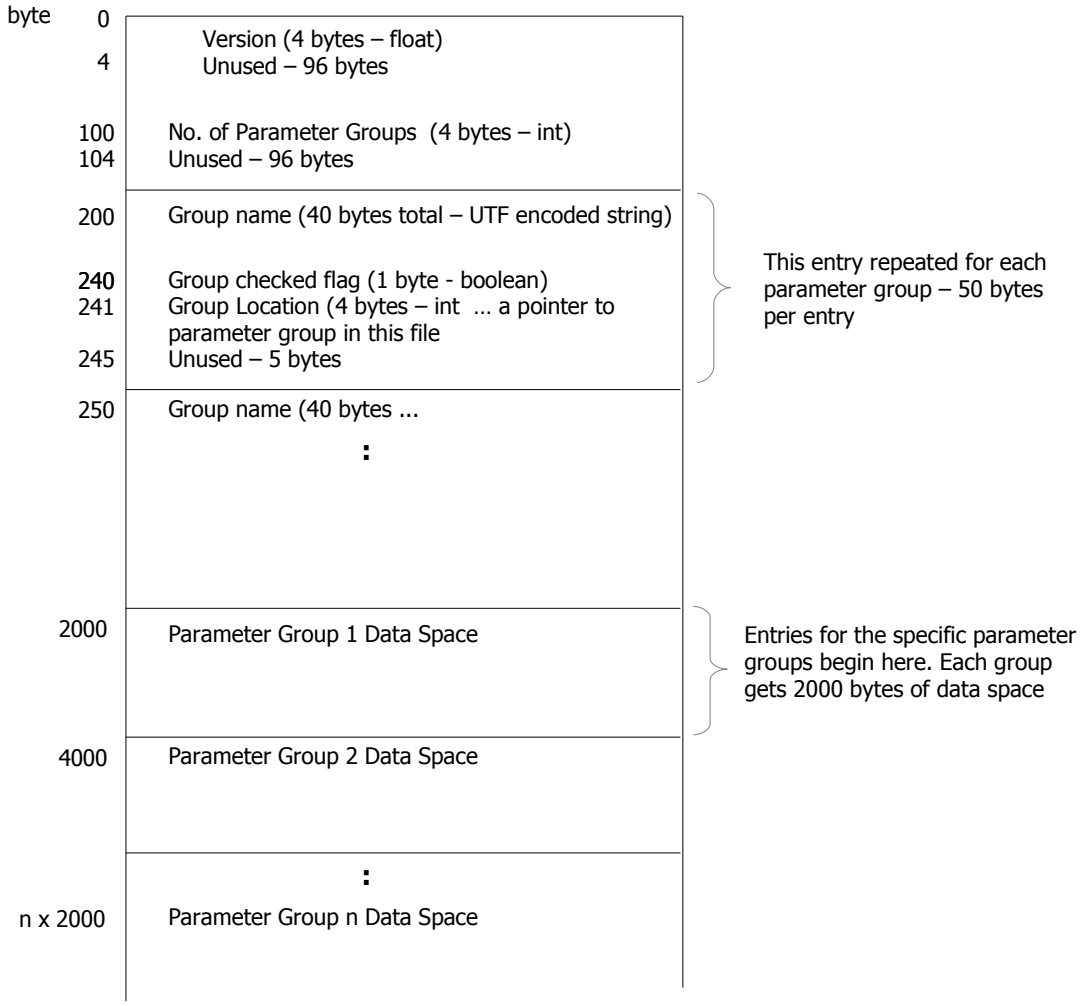


*Figure 4.4: Structure of the simulation parameter file*

#### *4.2.2.2 Simulation Parameter Group*

There are currently six (6) simulation parameters that are stored in the parameter file. The number of agents in the Red Team, number of agents in the Blue Team, the number of simulation runs to be iterated when the simulation engine is invoked, and the number of time-steps per simulation run. There are additionally two check-boxes and these are used to indicate whether the simulation engine will display a GUI-based representation of the world while the simulation is progressing, and are log-files to be kept.

If the *GUI Display* check-box is selected, then the simulation engine displays a window showing the agents and their location within the simulated world as the

simulation unfolds. This is a useful option when developing a scenario and viewing a single simulation run, but when a trial of 2000 simulation runs is being conducted it can be disabled by unchecking this parameter as it will slow down the simulation. Additionally, when the *Generate Log Files* option is checked the simulation engine will record extensive information from the the simulation so the different simulation runs can be analyzed at a later stage. The number of time-steps in the simulation and the number of simulation runs are used to drive the simulation engine which will be discussed in a later section. The Simulation Parameter input form can be seen in Figure 4.5.

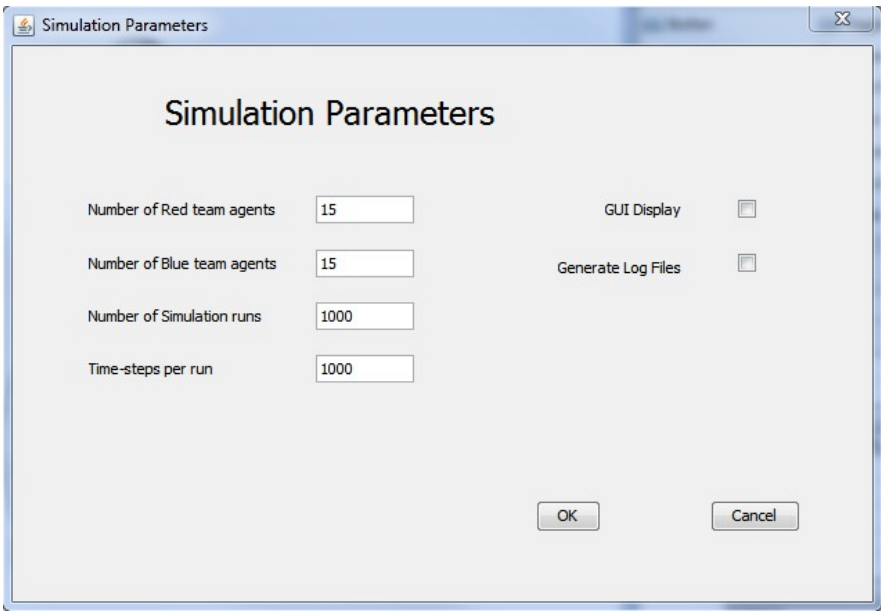


*Figure 4.5: Simulation parameter input form*

#### *4.2.2.3 Agent Parameter Group*

The *Agent parameter group* is used to store the initial location of all the agents in the simulation. Unlike either of the other two parameter groups, this parameter group relies in values set within both the *World Parameter group* and the *Simulation Parameter group*. The Agent Parameter input form is shown in Figure 4.6. When this form is initially loaded, the World parameters are located and the size of the simulated world is then used in creating an input form that matches the world size, see Figure 4.6. By using an input form which matches the simulated world in size, we can then use the

mouse pointer to click into the world and locate all the agents in the simulation into initial positions. These agent locations can then be saved and used when the simulation engine begins execution.

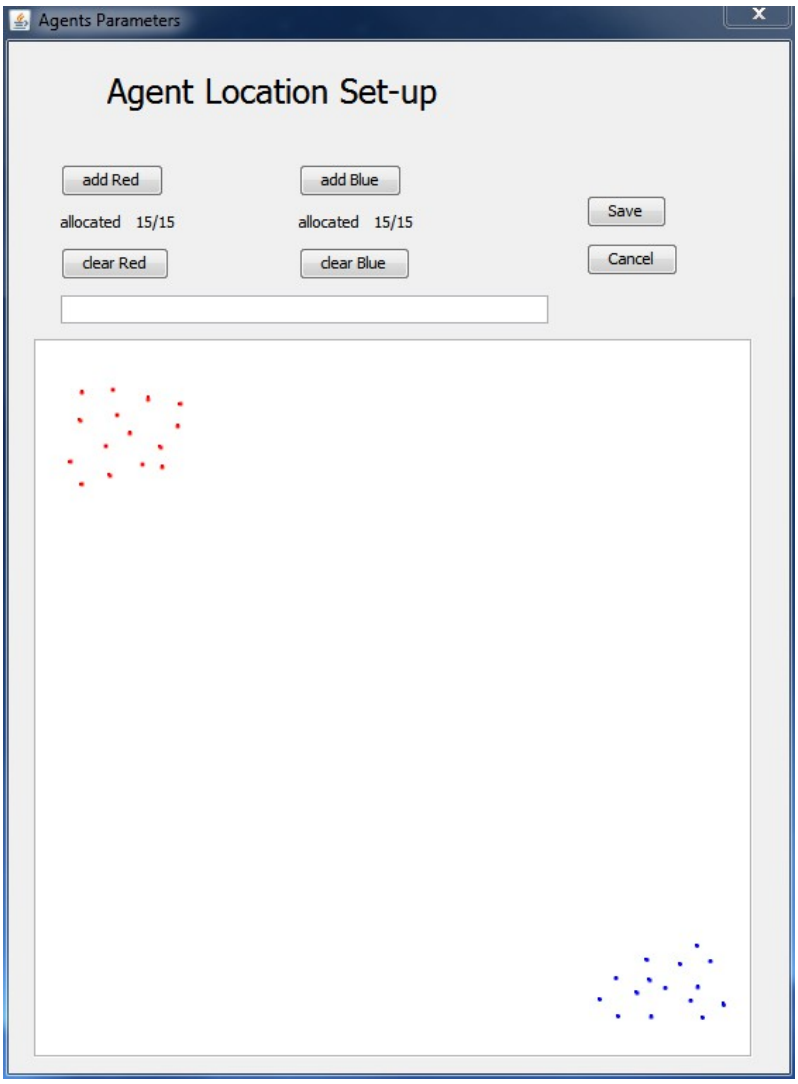


*Figure 4.6: Agent parameter group input form*

As the agents are being located, the *Agent Parameter* input procedure uses the values for the number of Red and Blue agents initially set in the *Simulation Parameter* input to keep track of how many agents have been located. You cannot allocate more Red agents or Blue agents than were specified in the *Simulation Parameter Group*. This helps to maintain the integrity of the parameter groups, and the simulation engine can then rely on the error-checking of the parameter entry code. The Java code for the Agent Parameter entry can be seen in Appendix A.4.

## 4.3 The Simulation Engine

The Simulation Engine Java class is at the heart of the simulation tool used in this research. All other options on the Main Menu GUI of the simulation are to support the parameter group entry. When the *Simulation* option on the main menu is selected, the simulation tool creates an instance of the simulation engine and invokes it. The overall architecture of the simulation engine is shown in Figure 4.7, the code of the simulation engine class is listed in Appendix A.5.

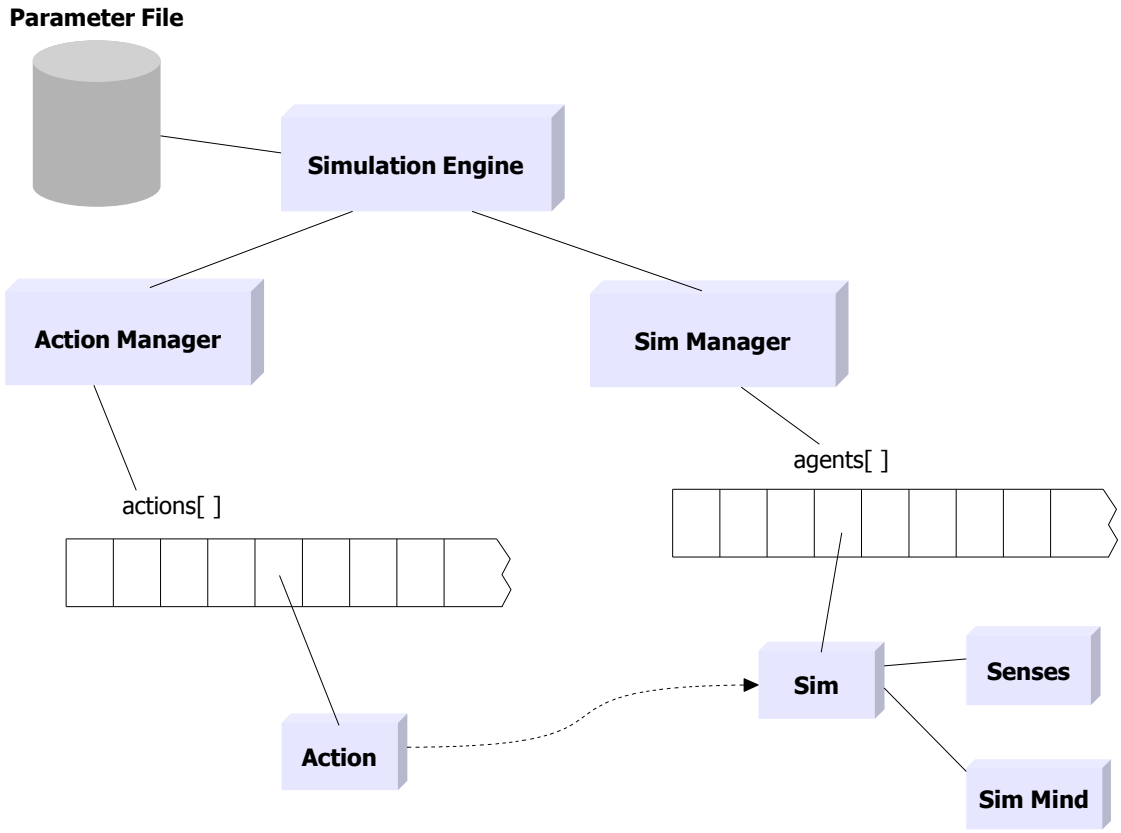


*Figure 4.7: Architecture of the simulation engine and components*

The simulation engine at its heart consists of an iterative loop which is governed by the *Number of Simulation Runs* and the *Time Steps per Run* parameters entered as part of the Simulation Parameter Group. This allows for a Simulation trial to be conducted where the *Number of Simulation Runs* parameter governs how many simulations will be conducted as part of the trial. Each simulation will be allowed to progress through the number of time steps indicated by the *Time Steps per Run*

parameter. This allows the researcher to set up trials with thousands of simulations, each running for hundreds of time steps which can be run without supervision. This can be a lengthy process and take a number of hours to complete. Figure 4.8 Shows a screen capture of such a trial in progress.

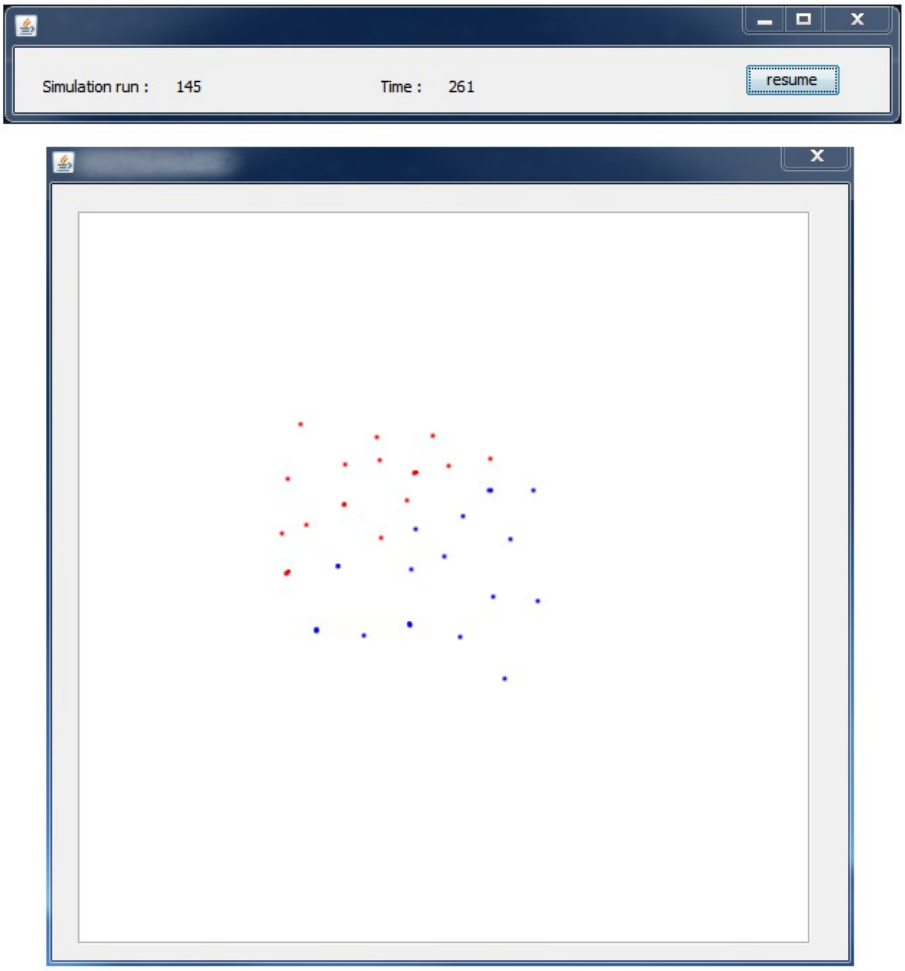


*Figure 4.8: Screen capture of a simulation trial in progress*

The simulation creates a status bar which is updated at each time step showing the current simulation run along with the current time within the simulation, see Figure 4.8. If the *GUI Display* check-box is selected in the *Simulation Parameter Group*, a visual display of the simulated world is displayed showing the location of each of the agents within the world. Each agent is shown as colored dot in the world, and the color of the agent indicates the team to which it belongs.

As indicated in Figure 4.7, the simulation engine additionally creates two other Java objects, the *Sim Manager* and the *Action Manager* objects. The purpose of the Sim Manager is to control and manage all the Sims throughout the simulation. The term Sim is used to refer to the agents of the simulation. This is a term borrowed from the suite of electronics games called Sim City[3], where Sims represented simulated entities within the game. The use of the sim manager alleviates the sim engine from having to deal with the micro-management of the sims throughout the simulation. The sim manager provides for controlling the thinking pattern during the simulation as well as handling sim deaths.

The purpose of the Action Manager is to control and manage all the actions created by the sims during their '*thinking*' phase at each time-step. The processing of the actions is initiated by the sim engine by calling the action managers *processActions*() method, see Appendix A.5. This in turn calls each of the actions *process*() method.

### 4.3.1 Sim Manager Architecture

An overview of the architecture of the Sim Manager is shown in Figure 4.7, and the code listing for this class can be viewed in Appendix A.6. When the sim manager object is first created, it uses the parameters in the *Simulation Parameter Group* as well as the parameters in the *Agent Parameter Group* to initially create an array called *agents[]*, large enough to hold an entry for each agent in the simulation. For each agent in the simulation, the sim manager then creates a sim object. A Sim represents the physical aspects of a simulated agent. This object contains elements which record all information necessary on the physical representation of the agent in the world, such as size, location, health, vision range, direction and others. The code for this class is provided in Appendix A.7.

The Sim object in turn creates two other object, a *Senses* object and a *SimMind* object. The Senses object represents the senses that the physical Sim has available to it, currently only vision is implemented and the Sim can use this to 'look' at the world and perceive other Sims in proximity to itself. The SimMind object is an abstract Java class that represents the cognitive aspect of the sim, or the sims mind. The code for the Java

3 Sim City is a product of Maxis, a subsidiary of Electronic Arts corporation.

SimMind object can be seen in Appendix A.8. This class is called when it is the corresponding sims turn to '*think*' and decide on a course of action. A representation of these classes and their association to each other is shown in Figure 4.9.

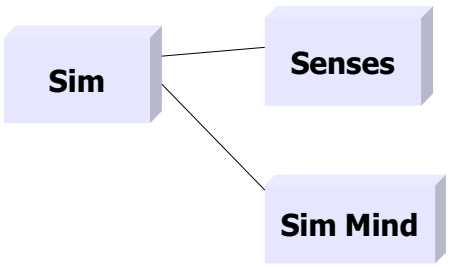


*Figure 4.9: Representation of the Sim classes which model an agent in the simulation*

A sim object is created for every agent that will be represented in the simulation. The sim manager then stores each of these sim objects in the *agents[]* array as indicated in Figure 4.7. This array is then used within the processing loop of the sim engine when each sim is given a turn to '*think*', and additionally the array is used when updating the results of the sims actions.

### 4.3.2 Action Manager Architecture

Similar to the Sim Manager, an overview of the architecture of the Action Manager can be seen in Figure 4.7. When the Action Manager object is first created it creates an ArrayList called *actions[]*, which is an extensible Java array capable of holding action objects. The Java code for this object can be seen in Appendix A.9. This array is not initially populated, instead the actions are created by the sim objects during their *thinking* cycle when they decide on a course of action. The sim objects then store these action objects in this extensible array for later processing. The code for the Java action object can be seen in Appendix A.10. The action object is an abstract class, of which there are a number of implementations, one for each possible action that an agent can perform. The range of possible actions an agent can perform are discussed in detail in the next Chapter, Chapter 5 Simulation Results. Each action is related to a sim (as the originator of the action), and this is indicated by the dotted line between Action and Sim in Figure 4.7.

The Action Manager object has one method called *ProcessActions()*. This is called by the simulation engine after every agent has had a chance to think and decide on an action to perform. When the ProcessActions() method is called, before application of the actions in the *actions[]* array is undertaken, it is first randomized.. This is an important step often neglected, as it helps remove any bias towards specific agents in the simulation due to their positioning in the *agents[]* array. When all actions have been processed, the *actions[]* array is cleared in preparation for the next time-step iteration in the simulation.

## 4.4 Representation of Time

In distillations such as the one that this research describes, it is possible to model the agents operating in real-time. By using Object Oriented techniques coupled with multi-threaded programming such as that available in Java, each agent can be given its own independent execution path.

Implementing each agent as a separate object and instantiating them as a separate thread allows us to model real continuous time within the simulation by having the agents operate under their own control in their own time. However there are practical limitations to this which must be considered, not the least being the uncertain nature of the Java Virtual Machine (JVM) in the processing of multi-threaded applications. The nature of the multi-threading in the Java Virtual Machine has been documented in many articles, eg. (Lee, 2006) (Darbyshire, 1998) and a problem exists for implementing agents using a multi-threaded paradigm. A simulation should be constructed to give stable results despite the implementation of threads, but the Java Virtual Machine delegates the scheduling of threads to the underlying operating system. Thus a simulation running on different operating systems may report different results due to a difference in scheduling algorithms. Additionally, there is no guarantee of the order that the threads will executed, or even if over long simulation run that there would be a fair distribution of computer time allocated evenly among the threads. This has the possibility of skewing the results by an agent performing poorly in such a simulation

solely because the underlying thread scheduling did not allocate a fair amount of Central Processing Unit (CPU) time. Additionally, the complexity of the code and the uncertainty of scaling such an implementation to large numbers of agents can make this an uncertain implementation.

Instead, by utilizing a discrete time-step simulation, we can achieve good results without the uncertainty associated with the JVM's multi-thread processing. With a discrete-time simulation, time moves forward in the simulation in evenly spaced steps, with certain defined operations occurring between each step. Such an implementation has far greater control over the allocation of time and other resources within the simulation and can assure a fair distribution of time allocated to each agent participating in the simulation. Within the simulation being constructed for this research, time is an abstract quantity which is represented by the passage of iteration cycles in the simulation engine (refer to Figure 4.7).

```
void runSimulation() {
     int currRun, time;
     boolean finishedRun;

     for (currRun = 1; currRun <= simRuns; currRun++) {  // iterate through all the sim runs
            // need to reset the simulation each time it re-starts ???
       time = 1;
       finishedRun = false;
       bar.setRun(currRun);

       while (!finishedRun) {  // now perform the current simulation run
         bar.setTime(time);
         simMan.cognition();       // all agents do their thinking and assemble their actions
         actMan.processActions();  // now process all the sims actions

         time++;
         finishedRun = isFinished(time);
       }

    }
  }
```

*Listing 4.1: Simulation engine main loop*

Listing 4.1 shows the code for the main iteration loop of the simulation engine. The complete code for this Java class can be seen in Appendix A.5. As can be seen in

Listing 4.1, time is incremented once for each iteration of the main loop. Thus time is an abstract quantity that does not directly relate to real time, but is used as a measure of progress through the simulation. However, by progressing through the simulation in this manner we can better control what each agent can do within these virtual *clock ticks*.

Each iteration of the simulation engines main loop then represents one time cycle, or one clock tick. Within this period of abstract time, two object methods are called, *simMan.cognition()* and *actMan.processActions()*. The *simMan.cognition()* method is invoked as part of the Sim Manager class, see Appendx A.6. This method is shown in Listing 4.2, and its function is to scan through the array of agents taking part in the simulation and call the *think()* method of the *mind* object associated with each agent.

```
void cognition() {
    int i;
    for (i = 0; i < agents.length; i++) {
        agents[i].mind.think();
    }
}
```

*Listing 4.2: Sim Manager cognition loop*

The mind object of each agent is an implementation of the *Sim Mind* abstract class shown in Figure 4.9 and represents the cognitive aspect of the agent. Each implementation of *Sim Mind* has a *think*() method which is called by the Sim Manager in Listing 4.2 and this represents the thinking process (or *thinking cycle*) that each agent goes through each time cycle. The *thinking cycle* will be discussed in more detail in the next section, but during this cycle, each agent senses the world and determines what action it will undertake. The appropriate action is created as a separate object and stored in the *action[]* array. When all agents have determined their action, the simulation engine then calls the *processActions*() method of the action manager object (*ActMan*). This method processes all the actions created by all the agents by scanning through the *actions[]* array and applying all the actions. However the *actions[]* array is first randomized to ensure fairness in the simulation.

Once the simulation engine performs these two tasks, time is incremented and the process is repeated until either a specific number of time cycles have passed or all the agents on one team have been eliminated. If an agent dies as a result of the application of an action then it is removed from the *agents[]* array and takes no further part in the simulation.

## 4.5 The Thinking Cycle

As indicated in the previous section, the thinking cycle for each agent is initiated by the simulation engine once for each time cycle. During this loop, each agent has the *think()* method of their implementation of the *Sim Mind* object invoked. This allows the agent thinking time, in which it perceives the world, determines its state and then chooses an action to perform. The actions for each agent are stored for later processing in the simulations *actions[]* array. In this manner, each agent is then allowed to complete its thinking cycle before any of the actions are processed, thus not allowing actions determined by other agents to interfere with the agents thought processes during this time cycle. Essentially, all agents are able to make their determinations based on the same world state. All actions are processed after all agents have had their *think()* method invoked. This is depicted in Figure 4.10.

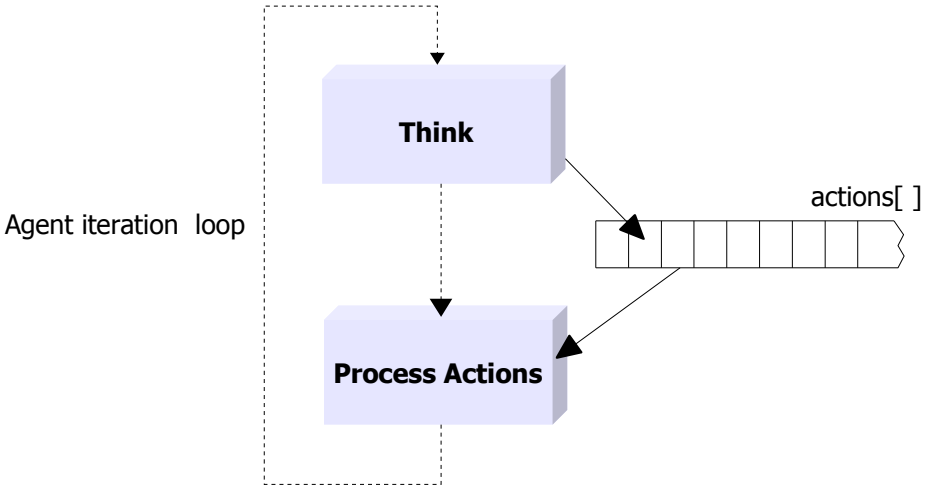


*Figure 4.10: Agent iteration loop*

As previously indicated, the *actions*[] array is randomized before actions are applied, performing this task eliminates any inherent unfairness caused by the order of

the actions, which in turn is determined by the order that the agents *think()* method is invoked. As the actions are processed after all the agents have gone through their thinking cycle, and the actions are randomized, we can let the simulation invoke all the agents *think()* methods in a static order without biasing the simulation.

The agents are divided into two teams, *red team* and *blue team*. These are discussed in length in Chapter 5 Simulation Results. The agents on the blue team represent the focus of this research and they have been constructed using a Reinforcement Learning algorithm which governs their actions. In the simulation, these agents utilize the *BlueMind* Java class to implement a concrete version of the *SimMind* abstract class. The code for the *BlueMind* Java class is listed in Appendix A.11. The *think()* method for the *BlueMind* class which is invoked by the simulation engine during each time cycle iteration is shown in Listing 4.3.

```
public void think() {

   scan();
   oldState = newState;
   oldAction = newAction;
   newState = getCurrentState();
   reward = stsMon.getReward();
   if (reward != 0) {
         if (reward > 0) sendComms(oldState, oldAction, reward);
   }

   if (oldState != -1) { // not first time around
         q[oldState].values[oldAction] = q[oldState].values[oldAction] +
                  stepSize * (reward + discountRate*maxActions(q[newState]) -
q[oldState].values[oldAction]);
   }

   // check comms for team members experience before choosing an action
   receiveComms();

   newAction = chooseNewAction(newState);
   takeAction(newState, newAction);
}
```

*Listing 4.3: Reinforcement Learning agent think() method*

It can be seen that the *think()* method shown in Listing 4.3 implements the Q-learning update function from Sutton and Barto (Sutton and Barto, 1998). The update

function is discussed in detail in *Chapter 2 Literature Review*, and shown in Equation 2.1. It is additionally reproduced again below in Equation (4.1):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \qquad (4.1)$$

The Q-learning update function is at the heart of the Q-learning algorithm used by agents implementing the *BuleMind* Java class, and its functioning is explained in detail in *Chapter 2 Literature Review* and in further detail in *Chapter 5 Simulation Results*. The statement from Listing 4.3 implementing the update function is

```
q[oldState].values[oldAction] =
q[oldState].values[oldAction] + stepSize * (reward + discountRate*maxActions(q[newState])
```

In order to implement the Q-Learning update function for a particular agent, the simulation must be able to determine, the current state the agent is in, as well as the action the agent has just taken, along with the previous state and actions. This is accomplished with the variables:

```
int oldState = -1, newState = -1; // to keep track of the old and new states
int oldAction, newAction;         // keeps track of the old and new actions
```

which can be see in Appendix A.11. The entire algorithm is driven by the data stored in each agents *state-action* matrix, represented by *Q*, in the Sutton & Barto Q-Learning update function (Sutton and Barto, 1998), shown in Equation (4.1) above. The state-action matrix allows the algorithm to store individual double-precision floating point numbers for each possible action that can be performed in each of the *States* that and agent can find themselves. Thus, *Q*, the *state-action* matrix, is a representation of the *state-space* of the environment for the reinforcement learning algorithms. The *state-space* for the research problem is discussed in detail in *Chapter 5 Simulation Results*. However, due to the nature of the state-space, not all actions are possible in all states, thus is is difficult to implement *Q* and a standard matrix for the purposes of computation.

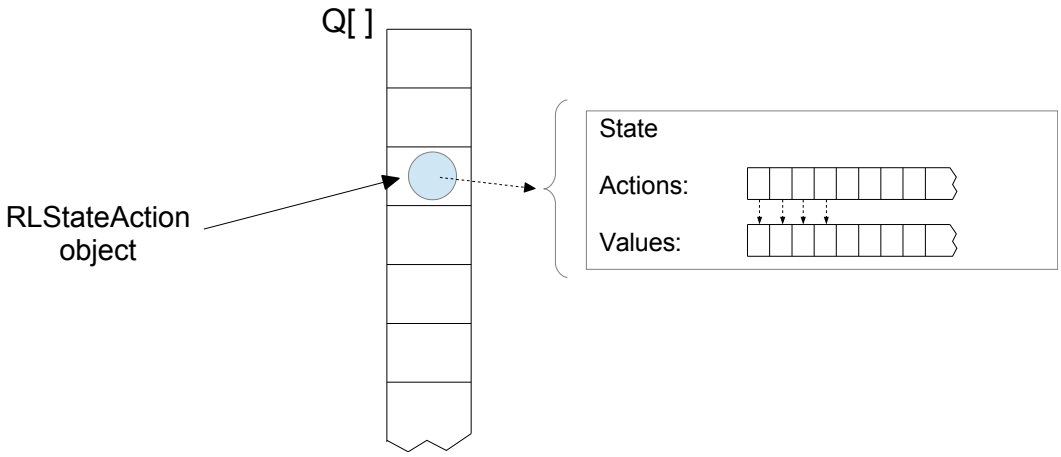


*Figure 4.11: Implementation of Q, state-action matrix*

*Q* is implemented as an Array of *RLStateAction* objects, and each of these objects in turn uses an *actions*[] array and a *values*[] array. The *actions*[] (not to be confused with the actions to be performed array represented in Figure 4.10) is an array of strings to hold the names of the possible actions of the current state. The *values*[] array holds the corresponding calculated values of the update function for the individual actions. This is depicted in Figure 4.11. The code for the *RLStateAction* object can be seen in Appendix A.12, along with the corresponding implementation of the attributes as shown i n Figure 4.11. Essentially, the learning of the agent is the accumulation of double precision floating point values in the *values*[] array, where each *value*[i] corresponds to a specific action *action*[i] of the current state. The number in *value*[i] is accumulated based on the reward received and application of the update function in Equation 1. Thus the higher the value stored in *value*[i] the more likely that in a particular state, the corresponding action *action*[i] will result in a positive reward.

After the Q-Learning update function has been performed based on the reward received from the environment, the agent chooses a new action to take through the method call to *chooseNewAction(newState)*, based on the current state. This will normally be chosen by selecting the array element from *values*[] with the highest recorded value, then taking the corresponding action from the *actions*[] array. This action is then recorded by the call to *takeAction(newState, newAction)*. As previously discussed, the action is not executed immediately but stored in an array for processing once all agents actions have been determined as shown in Figure 4.10. The thinking

cycle for the current agent in this time period then ends and the thinking cycle for the next agent in the list is invoked. After all agents have completed the thinking cycle, the accumulated actions are processed and the thinking cycles invoked again.

Within the *think*() method, communications are are processed, but these will be discussed in a later section.

## 4.6 Representation of State

In the previous section, the *RLStateAction* object identifies itself by implementing the *RLState* object as an attribute. See the following line from the *RLStateAction* object code from Appendix A.12.

```
public RLState state;      // represents a 'State' an agent can be in
```

The code for the *RLState* object can be seen in Appendix A.13. This object defines seven (7) variables used to partition the State Space for the environment, based on the agents own health and the types and numbers of other agents in sensory range. The partitioning of the state space is discussed in detail in *Chapter 5 Simulation Results*, and is briefly presented here for completeness. Apart from the health attribute, the other attributes partitioning the state space are concerned with the agents sensory capability and the number and ratios of enemy agents to friendly agents within sensory boundaries. Each agent has a visual sense which enables it to locate all other agents within a circular area, with itself as the center and the length of the visual range as the radius.

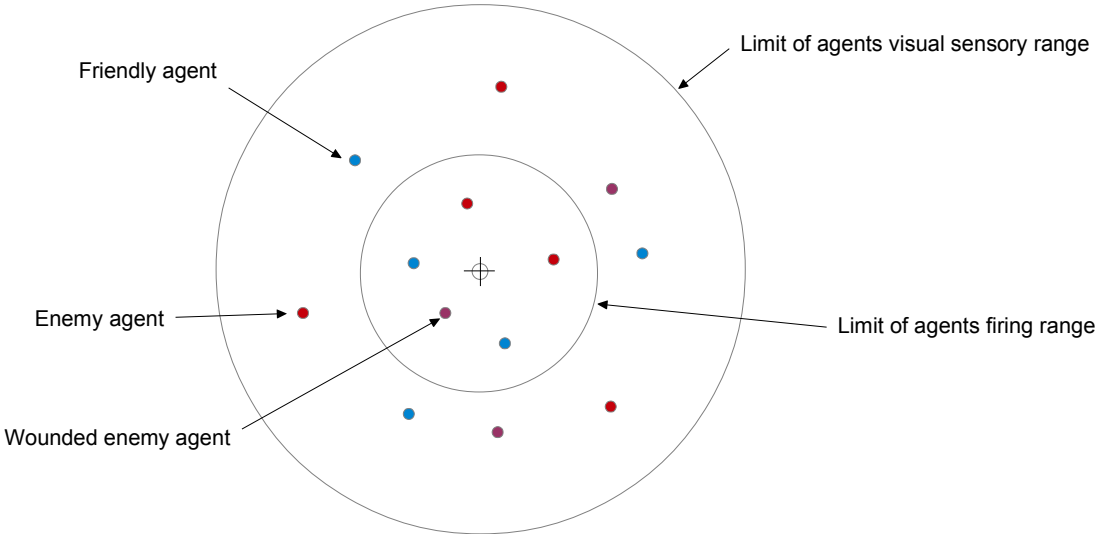


*Figure 4.12: Depiction of agents sensory ranges*

The range of an agents visual sense is set as a parameter to the simulation and is set as a distance which is relatively small compared to the distance from one end of the agents world to another. This ensures that agents only have a partial view of the world, which is more realistic for the underlying domain than giving agents a complete view of the simulated world. Each agent carries a weapon capable of firing one round per simulated time unit, and the range of this weapon is set to be a quantity less than the visual range of the agents. Thus the firing range of the agents weapon and the visual range of an agent are used as radii of two concentric circles for each agent. This is depicted in Figure 4.12. By using the two sensory ranges, visual range and firing range, we can utilize some simple metrics to help form the state-space for the agents.

The seven state-space metrics used are:

***EiVR*** Enemy in Visual Range two states 0 - No 1 - Yes

***WiVR*** Wounded in Visual Range two states 0 - No 1 - Yes

***EiFR*** Enemy in Firing Range two states 0 - No 1 - Yes

***WiFR*** Wounded in Firing Range two states 0 - No 1 - Yes

***health*** this is the quantized health state... currently has two states

0 (bad - percentage health <50%)

1 (good - percentage health >= 50%)

***visualEFRatio*** quantized value for quantity enemy/friendly in visual range

0 - more friendlies

1 - equal amounts

2 - more enemy

***firingEFRatio*** quantized value for quantity enemy/friendly in visual range

0 - more friendlies

1 - equal amounts

2 - more enemy

All of the above state-space metrics except the agents *health* are based on the visual and firing ranges. The nature of the simulation is such that these metrics represent a division of the state-space based on what is important to the agent in order for it to be able to achieve its goals. Of the seven metrics, two have three possible values, and five have two possible values, thus allowing for a state-space 288 possible states. However, we can reduce the state-space size by eliminating inconsistencies between values in the different state variables. The number of possible state spaces then reduces to 68, thus the size of the Q array in Figure 4.11 is 68. To aid in the initial set-up of the simulation each time it is opened, a text file called *StateActionMap* is maintained that records all possible states and all possible actions allowed in each state. This text file can be seen in Appendix B.1. Appendix A.14 shows the *RLUtil* simulation code that is executed each time the simulation is opened. This code reads and interprets the *StateActionMap* text file containing the 68 coded states along with allowable actions for each state and builds the initial *Q*[] array from scratch for each agent. This code is called from the *BlueMind* code listed in Appendix A.11. Figure 4.13 depicts building an agents state-action matrix, *Q*[], from the *StateActionMap*.

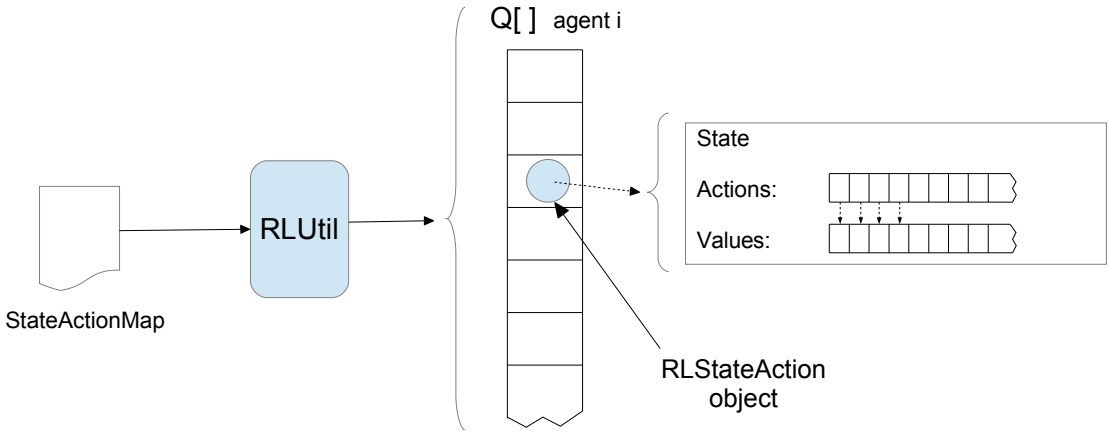


*Figure 4.13: Building the agents Q[ ] array from the StateActionMap*

Within each *RLStateAction* object in *Q*[] from Figure 4.13, the state is represented as a collection of the state variables, see *RLState* Object code listing in Appendix A.13, or as a 7-tuple in the *StateActionMap* in Appendix B.1. For instance, the 7-tuple (1 1 1 0 1 2 0) in the *StateActionMap* represents a state where the agent has enemy agents in visual range; has wounded enemy in visual range; has enemy agents in firing range; no wounded enemy in firing range; has good health; has more enemy agents than friendly agents in visual range; and has more friendly agents than enemy agents in firing range.

During the thinking cycle, each agent determines its current state by using its sensory capabilities and calling the *scan*() method, then calling the *getCurrentState*() method. This uses the data from scanning the environment, builds an *RLState* object and then compares it to the corresponding states in the *Q*[] array until it finds a match. Once a match is found, we then have access to the list of possible actions and their corresponding values for that state. The Q-learning algorithm shown in Equation (4.1) can then be applied.

## 4.7 Communication

In the previous sections we discussed the Q-Learning update function executed by each agent during their thinking cycle. The Q-Learning update function shown in Equation (4.1) from (Sutton and Barto, 1998), while utilized during the thinking cycle, is not the only reinforcement learning update being performed by each agent. The aim

of this research is to study the effects of communication in a multi-agent simulation which utilized reinforcement learning. The agents communicate their experiences with their team members during the simulation and pass on their own learning experiences. This is discussed in detail in Chapter 5 Simulation Results, and in fact forms the focus of the research conducted in this thesis. In order for agents to be able to share their learning experiences and indeed, learn more from the other agents sharing, the agents need to be able to communicate. Additionally, the Q-Learning update function needs to be modified in order to not only process an agents rewards, but also take into account the learning experiences from other agents, to enhance its own learning from these experiences.

The *BlueMind* Java object whose code is listed in Appendix A.11, in reality, implements an enhanced version of the Q-Learning update function from Equation (4.1). The *BlueMind* code implements the multi-agent Q-Learning update function developed for this research and shown in Equation 5.2. This enhanced update function is the subject of study in *Chapter 5 Simulation Results*, and will not be detailed here. However the *BlueMind think*() method utilizes communication in order to send and receive learning events from one agent to another in the same team.

The Q-Learning update function has been moved into a separate method called *ProcessReward*() as shown in Listing 4.4. By parameterizing the updating function it can be called more than once quite easily which is what is required for the enhanced multi-agent Q-Learning update function.

```
/*  This method processes the reward received from the system.  Currently
      uses equation 6.6 from Sutton & Barto for Q-Learning

      required parameters
      state  - state of system when action was chosen
      action - action chosen that leads to this reward
      r      - the reward gained from State- Action pair
      nstate - the new state resulting from the application of the action
  */
  private void processReward(int state, int action, double r, int nstate) {

    // update the action-value matrix
    q[state].values[action] = q[state].values[action] +
            stepSize * (r + discountRate*maxActions(q[nstate])
                    - q[state].values[action]);
  }
```

*Listing 4.4: ProcessReward() method listing*

As can be seen in the BlueMind code listing in Appendix A.11, the think() method attempts to retrieve a reward the agent might have received from the previous time step based on the action they performed. Then if there was a reward, it performs the following overall steps:

- send communication of reward to all team members
- process the reward (perform update function)
- check for incoming communication from other agents
- choose a new action to perform
- take that action

The sending and receiving of communication is actually implemented in the enhanced Q-Learning update function from Equation 5.2. The *sendComms*() method builds a tokenized string consisting of the state the agent was in and the action that was taken to generate the reward, the reward, the new state that resulted from the action and the identity of the agent that is sending the message. For example, such a message string generated from a particular simulation run is shown in the line below:

*State 5 Action 1 Reward -0.5 newState 11 from 24*

This message indicates that a reward of (*-0.5*) was received from agent *24* when agent *24* performed action *1* while in state *5*. This resulted in agent *24*'s new state being *11*. In this message, the agent number is an index into the simulations *agent*[ ] array, which contains one entry for every agent in the simulation. Having this index gives the simulation access to the agent object. The state references, *11* and *5*, refer to an index into an agents *Q*[ ] array as shown in Figure 4.13, giving us access to the *RLStateAction* object representing that particular state. The action number then, is an index into the *actions*[ ] array of the *RLStateAction* object in Figure 4.13, which represents the action taken, giving us in turn, access to the corresponding *values*[ ] entry for that state, for that agent. One of these message strings is sent to every agent in the same team as the agent that received the reward.

After the *think*() method processes the reward the agent received from its previous state-action combination, it then checks for any received communications. Any communications received will be of the same tokenized string message shown above, and are processed through the *receiveComms*() method. Each agent could potentially have received *n-1* messages where *n* represents the number of agents on the same team. Each of these messages in turn is deconstructed, and each message contains enough information to again perform the Q-Learning update function in Listing 4.4 as if the information was generated by the agent itself. All the messages are processed in this manner, thus adding the learning experiences of the other agents to the current agents state-action matrix.

A communication is sent from one agent to another as an action. We have previously discussed actions and stated that during each iteration of the simulation loop, representing one simulation time period, an agent can only perform one action. Although a communication is treated as an action for the purposes of the simulation coding, it is not regarded as an action as far as the simulation operation is concerned. This is consistent with the underlying domain as an agent may be performing an action and possibly communicating at the same time. When an agent constructs a tokenized message as the one shown above, it is sent to every member of the agents team, and thus generates *n-1* communication actions,where *n* is the number of agents on the team.

These communication actions are stored in the *actions*[ ] array to be processed at the end of the simulation loop after all agents have completed their thinking cycles. This is depicted in Figure 4.14

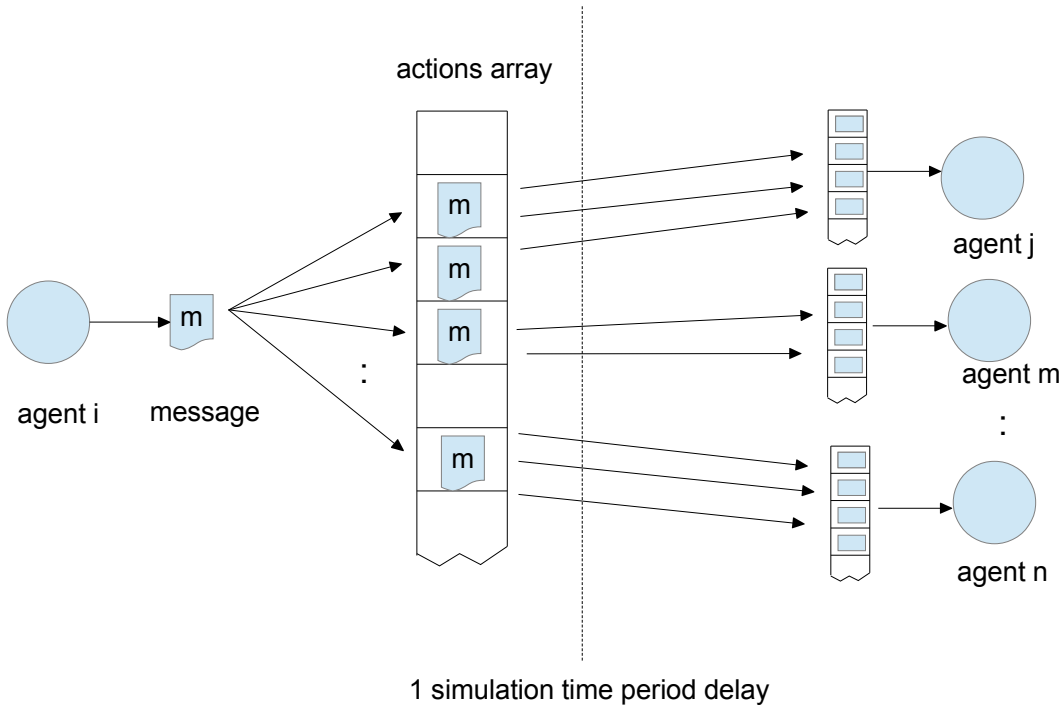


*Figure 4.14: Communications sent as actions in the simulation*

At the end of the simulation loop for the current time period, the *actions*[ ] array is randomized and all actions are processed. As a communication action is found it is stored in a *messages*[ ] array for the agent to which it has been addressed. As many agents may have had learning experiences and sent tokenized messages, each agent may have accumulated many messages during the processing of the communication actions. Although the *actions*[ ] array is randomized, the order of the tokenized messages for a particular agent is irrelevant as they are used to update the agents state-action matrix. In the next iteration of the simulation loop during its thinking cycle, an agent will then retrieve each of the messages through the *receiveComms*() method. After performing the Q-Learning update function from its own reward, each of the messages will be further used to update the agents state-action matrix. See Figure 4.14 for a depiction of this.

## 4.8 Implementation of an Agents Weapon

As a final note in discussing the simulation tool it is worthwhile discussing the nature of the weapon used within the simulation. The purpose of the simulation is to enable study of a complex adaptive multi-agent system, representing a hostile environment. It provides for a representation of an underlying domain of a battlefield. Thus each agent is provided a weapon which is used in the implementation of some of the actions an agent can perform. The weapon is an abstract weapon which enables an agent to fire a single shot/ projectile at another agent.

The weapon is simply implemented as a limited range weapon where the firing range is set at a distance less than the visual range of the agent. This is depicted in Figure 4.12, and assumes a 360 degree possible limited range. This is consistent with an agent being able to turn in any direction and fire. If used, the agent can fire a single shot within each of the actions that involve using the weapon. The projectile assumes a line-of-sight requirement. That is, an enemy agent can only be targeted if there is a line-of-sight from the agent performing the action to the agent being targeted. Additionally, the agent being targeted must be within firing range. During the action selection process, each agent has access to the distance from itself to all other agents within visual range. While this information is not directly representative of information available to agents in the underlying domain, it is not inconsistent as natural perception and training would give agents an approximate notion of distance.

The targeting of an enemy agent does not guarantee a hit, but a probability measure is used to determine if a hit is registered. This is based on distance to the targeted agent and a small random bias, consistent with the underlying domain. During the processing of the actions at the end of the simulation loop, if a hit is registered against an agent, its health points are decremented. Each agent begins with 100 health points and this is decremented by 15 each time a hit is registered against the agent. When the health gets decremented to a figure below 0, the agent is flagged as dead and takes no further part in the simulation. No allowance is made for a limitation on the number of rounds or projectiles that can be fired, so in effect the agent is deemed to have unlimited ammunition.

The weapon can be considered as an abstract implementation of a limited range rifle.

## 4.9 Control Group Description

In this section we give a detailed description of the control group of agents, referred to as the *Red Team*. The color designation is given on the basis that the color *Red* is used to indicate the agents of the control group on the output display, while the color *Blue* is used to represent the group of agents governed by the modified reinforcement learning algorithm which forms the basis of this research.

The agents in the control group represent a highly trained team and the behavior of this team is governed by a rigid control paradigm consisting of a series of weighted values for various behavioral aspects. Each agent will have a number of behavior variables that determine how it will react in any given situation. For example, willingness to engage; willingness to explore; tendency to obey orders etc. Each of these behavioral quantities is pre-assigned a weighted value before the simulation begins, which can be stored and retrieved for further simulation trials. The input screen for the weighted values affecting the control group behavior is shown in Figure 4.15. As can be seen in Figure 4.15, most of the weighted values are assigned as numbers between zero and one, though some values are assigned between negative one and positive one. These weights are used by the governing algorithm to determine each agent's behavior at each simulation time step and represent a probability which determines the agents action in a given situation. This is a typical agent control paradigm prevalent in such distillations, where the weights can be modified to study the outcomes, given different types of expected behavior in the agents.

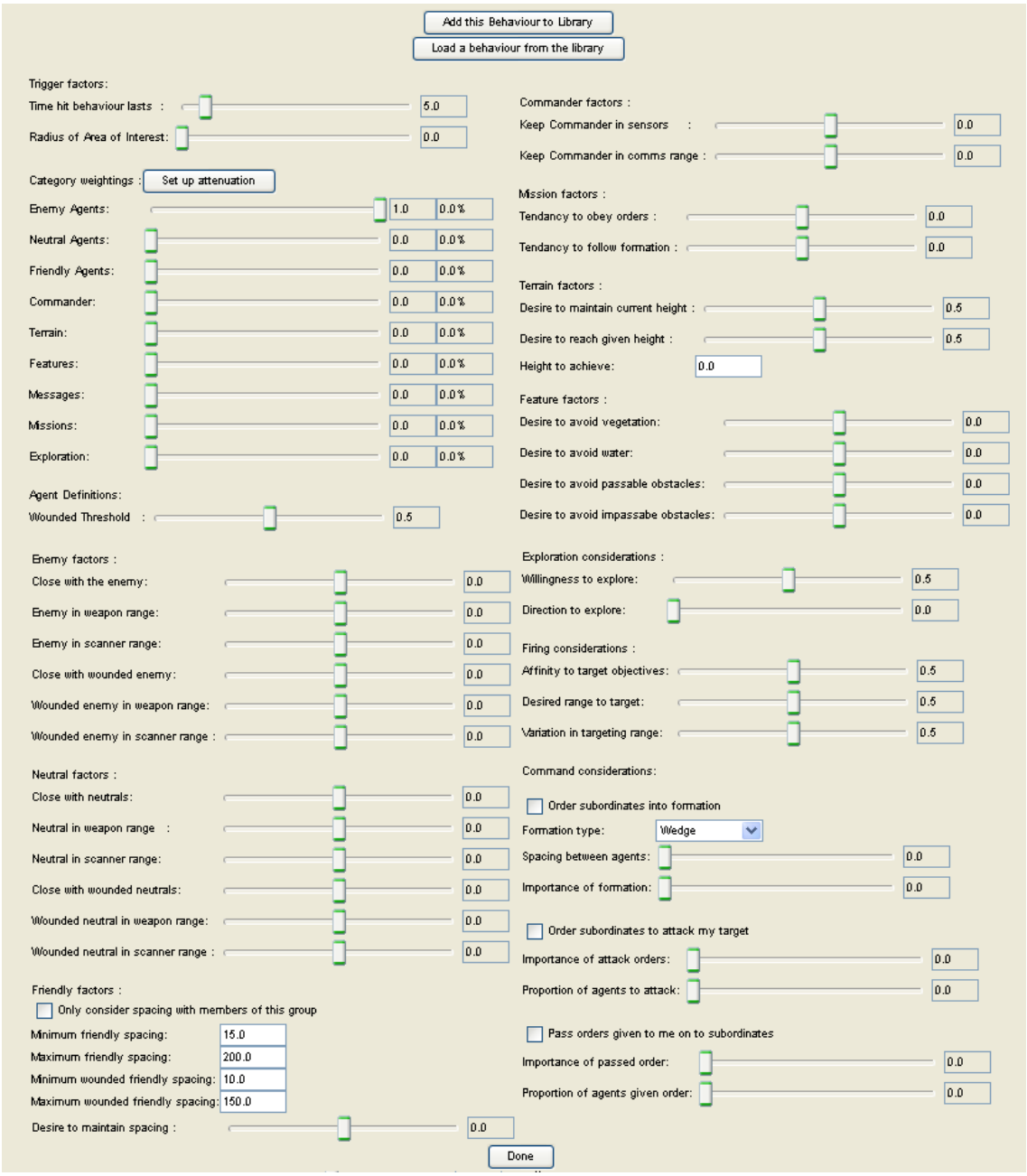


*Figure 4.15: Control Group weighted behavioral input screen*

As a further example, a small section of the control group weighted input screen from Figure 4.15 is shown in Figure 4.16. This section of the input screen shows values in the range of zero to one that are used to determine the behavior of an agent from the control group in relation to a number of variables including: proximity of the enemy and other neutral agents; enemy agents in sensor range; enemy agents in weapons range;

proximity of wounded enemy agents; proximity of neutral wounded in weapons and sensor range.

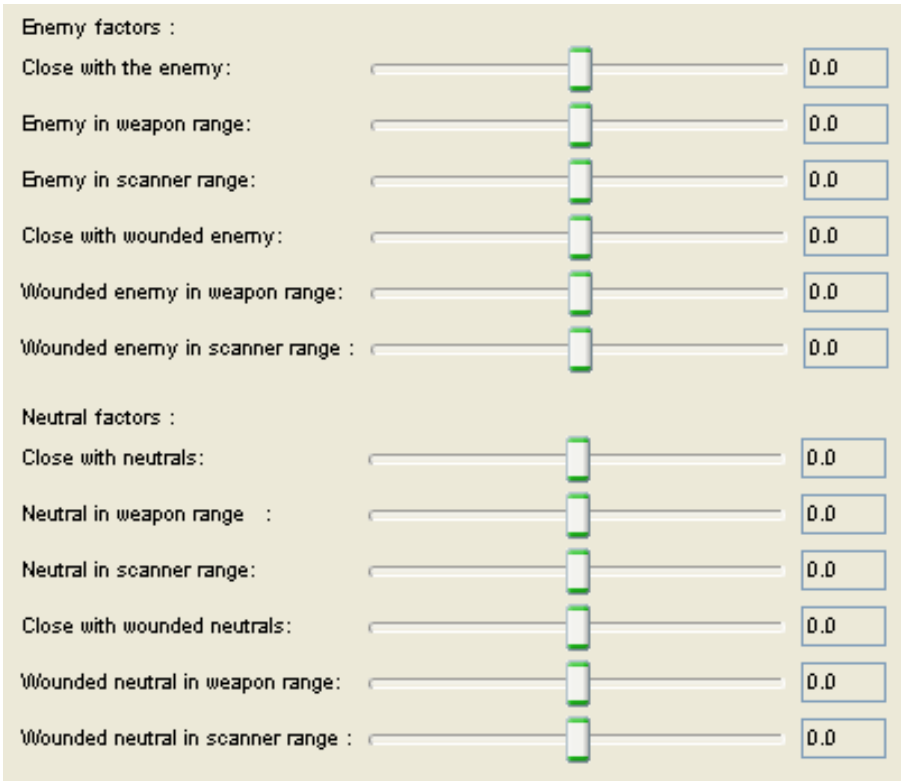


*Figure 4.16: Control group weighted input screen section*

At each time-step in the simulation, agents of the control group determine their actions based on probabilities determined from the weighted values associated with behavioral traits from the group (as shown in Figure 4.15). This behavioral determination is often referred to as instinctual behavior. This is because, unlike the group of agents controlled by the reinforcement learning algorithm discussed in previous sections, the control group behavior is pre-determined based on the weighted values. There is no variation from this behavior as the simulation unfolds and the control group can be thought of as a highly trained group acting on instinct gained through training.

The control group agents are not the focus of this research and are thus considered part of the environment from the perspective of the reinforcement learning agents.

## 4.10 Simulation Output

In order to study the effects of the agent communication on the outcomes of the simulation, and indeed to explore the operation of the simulation in detail we need a mechanism to view the effects of the agents actions on other agents as the simulation unfolds. The simulation has been constructed so that it has the ability to generate log files capturing specific data as the simulation progresses. Most of the data is captured by saving data at the end of each iteration of the simulation loop. The simulation produces four different output files to enabled a detailed study of the effects of communication and of the actions of each agent. These are

- crt files: records the state of each team at the end of each iteration of the main simulation loop
- cre files: this file captures relevant information about a 'Hit', damage inflicted to another agent as a result of an agent action, gathered during the processing of the *actions*[ ] array after all the agents have completed their thinking cycles
- actval files: records the state of each agents *Q*[ ] array at the end of an entire simulation run
- screen files: shows information displayed to the screen such as rewards found and messages sent to other agents during the simulation run

The information in these file enables the detailed analysis of the simulation runs that is provided in *Chapter 5 Simulation Results*.

A simulation trial will consist of a number simulation runs, and the above four files are produced automatically for each run. The files are named:

- xxxcrt.csv
- xxxcre.csv
- xxxactval.txt
- xxxscreen.txt

where xxx represents the number of the simulation run within the simulation trial. The format of the crt file can be seen in the sample below in Listing 4.5. An extract of the

file is provided in Appendix B.2. The first column represents the simulation time, the following columns represent, the team identification number, the total remaining health points of the team, the remaining number of agents in the team, the opponent team identification, the remaining health points of the opponent team and the number of agents left alive in the opponent team.

```
93  0   1500  15  1   1500  15
94  0   1500  15  1   1500  15
95  0   1500  15  1   1500  15
96  0   1500  15  1   1500  15
97  0   1500  15  1   1500  15
98  0   1485  15  1   1485  15
99  0   1485  15  1   1485  15
100 0   1485  15  1   1485  15
101 0   1485  15  1   1485  15
102 0   1470  15  1   1485  15
103 0   1455  15  1   1440  15
104 0   1440  15  1   1440  15
```

*Listing 4.5: Format of the crt file*

The crt file is a good way of tracking the overall performance of a particular team over the simulation, and this is explored in detail in *Chapter 5 Simulation Results*. The format of the cre file can be seen in the sample below in Listing 4.6, and an extract of the file for a particular simulation run can be seen in Appendix B.3.

The cre file captures information about agent 'Hit' events, when an agent causes damage to another agent. The structure if the cre file is as follows:

The first column shown is the agent identification of the agent being 'Hit', followed by its agent team identification, the identification of the attacking agent (needed to allocate rewards), the attackers team identification, the cause of the damage (currently only 1 is valid), the amount of damage in health points (usually 15), the simulation time, followed by the (*x,z,y*) coordinates of the agent being 'Hit'. The (*x,z,y*) coordinates are the coordinates of the agent in the simulated world, with the *z*

coordinate always 2.5 in the current simulation.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 11 | 0 | 1 | 15 | 97 | 257.8122977 | 2.5 | 232.0120181 |
| 27 | 1 | 4 | 0 | 1 | 15 | 97 | 336.7020277 | 2.5 | 333.1477133 |
| 4 | 0 | 28 | 1 | 1 | 15 | 101 | 267.2886271 | 2.5 | 244.9038441 |
| 4 | 0 | 27 | 1 | 1 | 15 | 102 | 269.6569372 | 2.5 | 248.1273682 |
| 27 | 1 | 15 | 1 | 1 | 15 | 102 | 334.5380405 | 2.5 | 329.7386828 |
| 22 | 1 | 4 | 0 | 1 | 15 | 102 | 330.3138412 | 2.5 | 361.5746663 |
| 26 | 1 | 29 | 1 | 1 | 15 | 102 | 323.7876139 | 2.5 | 342.4282157 |
| 4 | 0 | 26 | 1 | 1 | 15 | 103 | 272.0249382 | 2.5 | 251.3511192 |
| 4 | 0 | 27 | 1 | 1 | 15 | 106 | 279.1381997 | 2.5 | 261.0155633 |
| 21 | 1 | 19 | 1 | 1 | 15 | 106 | 352.3609714 | 2.5 | 334.0668851 |
| 4 | 0 | 28 | 1 | 1 | 15 | 107 | 281.5116504 | 2.5 | 264.2353042 |
| 27 | 1 | 15 | 1 | 1 | 15 | 107 | 332.3147524 | 2.5 | 326.3763337 |
| 18 | 1 | 4 | 0 | 1 | 15 | 107 | 349.0797507 | 2.5 | 359.7840282 |
| 4 | 0 | 29 | 1 | -1 | 15 | 110 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 29 | 1 | 1 | 15 | 110 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 28 | 1 | 1 | 15 | 111 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 27 | 1 | 1 | 15 | 111 | 288.6389437 | 2.5 | 273.8894065 |
| 8 | 0 | 7 | 0 | 1 | 15 | 111 | 250.0914752 | 2.5 | 264.4895332 |
| 11 | 0 | 6 | 0 | 1 | 15 | 111 | 282.1760662 | 2.5 | 258.3343467 |

*Listing 4.6: Format of the cre files*

The information in this file is invaluable in allowing a detailed analysis of the agents actions as the simulation unfolds.

The format of the actval file is shown in Listing 4.7, and an extract of the file for a particular simulation run is shown in Appendix B.4.

The actval text file is a textual dump of each agent's Q[ ] array (see Figure 4.13) at the end of the simulation run. The first line contains the identification number of the agent, every subsequent line contains the text 'state', followed by the state number (valid values are 0 -67)allowing for 68 states. The remaining columns are a dump of the values[ ] array for that agent (see Figure 4.13) where each value corresponds to allowed action for that state. Each state can have a different number of allowed actions so the remaining column numbers are not consistent. This file should be read in conjunction with the StateActionMap in Appendix B.1.

```
6
state 0 0.0  0.0
state 1 0.0  0.0
state 2 0.0  0.0  0.0  0.0
state 3 0.0  0.0  0.0  0.0
state 4 0.0  0.0  0.0  0.0
state 5 0.0  0.0  0.0  0.0
state 6 0.0  0.0  0.0  0.0
state 7 0.0  0.0  0.0  0.0
state 8 0.0  0.0  -0.049495000000000004  0.0  0.0  0.0  0.0
state 9 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 10 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 11 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 12 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 13 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 14 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 15 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 16 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 17 0.0  -0.095  0.0  0.0  0.0  0.096552154554  0.0
state 18 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 19 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 20 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 21 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 22 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 23 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 24 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 25 0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

*Listing 4.7: Format of the actval file*

The information in this file helps in an analysis of what the agents learned within each simulation run, and can be used to compare successful agents with non-successful agents. Finally, the structure of the screen file can be seen in Listing 4.8, and an extract of the file from a specific simulation run can be viewed in Appendix B.5.

This file is just the screen dump of messages sent to the screen during the simulation. This is used mainly as a counter check to ensure the simulation is running correctly and can be compared to the cre and crt files to ensure specific events coincide at the correct simulation times. An entry is sent to the screen every time an agent receives a reward, containing the simulation time it received the reward, the agent identification number, the amount of the reward and the simulation time the reward was actually generated.

Additionally, every time an agent sends a message to another agent, a slightly indented screen message is generated consisting of the simulation time, the agent

identification of the agent to whom the message is being sent, and the other parameters which enable the receiving agent to update its state-action matrix, such as the state and action identifications, the reward, the new state identification and the agent that sent the message.

```
102: 28: found reward = 0.5  from time 97.0
103: 15: found reward = 0.5  from time 100.0
  103: 15 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  103: 16 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  103: 17 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  103: 18 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  103: 19 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  103: 20 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  103: 21 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
103: 22: found reward = -0.5  from time 97.0
  103: 22 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
```

*Listing 4.8: Format of the screen text file*

# 5

# SIMULATION RESULTS

*There is nothing like looking, if you want to find something. You certainly usually find something, if you look, but it is not always quite the something you were after.*

– *J.R.R. Tolkien (The Hobbit)*

In the previous chapters we have presented relevant literature and related work as well as discussed in detail the design and structure of the simulation environment that was used to generate the results. The experiments were conducted with the original hypothesis in mind, that is, that allowing agents to communicate in a group situation where the group shares a common goal will improve the success chances of the group. While the existing literature well supports this hypothesis, the experiments were designed to further explore the deeper question of the exact nature of the improvements. A detailed analysis of the nature of the improvement in group success rates when communication takes place has not been adequately explored in the literature. The results presented here allow us to explore the fundamental relationship between agent communication and their success as a group.

In order to explore this question, a set of experiments were designed around a military style distillation which represents a battlefield simulation with two opposing forces. The opposing forces are represented by two groups of agents. In the control group each of the agents actions are controlled by a standard instinctual agent control paradigm, typical in such simulations. For each of these agents, the actions at each simulation step are determined by a set of weights associated with a number of

predetermined behaviors. Thus the agents whose actions are based on this control paradigm would be indicative of a group of highly instinctual entities acting on instinct rather than cognitive ability. The second group of agents is not controlled by a rigid control paradigm but rather by a reinforcement learning algorithm whose reward function is designed to achieve the desired end. This group of agents has no predetermined actions, and are initially untrained. The successful actions are learned by the agents using a combination of exploration and ε-greedy policies as the simulation progresses. Initial experiments compare the performance of the control group against the group whose actions are controlled by the reinforcement learning algorithm (tagged as the Blue Team). The control group is tagged as the Red Team.

In later experiments, the Blue Team members then communicate with each other as the simulation progresses and the results are then used to compare the performance of the Blue Team both with and without communication. Thus we can then observe the effects of the communication within the Blue Team and analyze the performance during the simulation and the surviving agents to provide a detailed picture of the deeper effects of the communication. In the following section we discuss learning events within the simulation and the nature of the communication that can take place between the agents.

## 5.1 Messaging and Learning Events

In order to study the effects of communication we need to consider carefully the basic nature of the communication that will take place in the system currently under study. The system which is at the center of this research is basically a military distillation which is used to study land combat a a conceptual level only. Hence it is a conceptual level battlefield simulation. As discussed in the previous chapter, Chapter 4, and in the introduction above, a Blue Team consisting of a group of agents controlled by a reinforcement learning algorithm is set against a control group in order to study the effects of the communication. Thus, the nature of the algorithm governing the Blue Team will determine the nature of the communication and additionally how and when the communication will take place within the simulation.

The type of Reinforcement Learning algorithm chosen to control the Blue Team agents is a Temporal Difference learning algorithm, specifically, Q-Learning. A Temporal Difference algorithm was chosen as this class of algorithm is the best representative of the way a team of actual agents would learn in the underlying domain which the simulation represents. In this domain, an actual group of combatants cannot run through a battle then restart thus learning from the experience. They must learn from experience as the simulation progresses, and either survive or die based on their performance. Hence the learning is crucial to their survival. The Temporal Difference algorithms allow an agent to learn from experience as the simulation unfolds. The Q-Learning algorithm was chosen as it is widely accepted and used in the literature, is relatively easy to implement and the agent does not need a model of the environment to begin.

The Q-Learning algorithm implemented is the one detailed in Sutton and Barto (Sutton and Barto, 1998) with the update function represented by Equation (5.1).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right] \quad (5.1)$$

The components of the above equation are given in the following explanation. The Q-Learning update function works by maintaining a 2-dimensional array ***Q*** whose size ***Q(S,A)*** is determined by ***S***, a set of possible states that an agent can be in, and ***A***, a set of possible actions an agent can choose from. Thus, $Q(s_i, a_j)$, $s_i \in S$ and $a_j \in A$ represents the state-action value pairs in Q and are decimal numbers used to help determine the optional action to take when in a particular state. A row in Q represents all possible action values for a particular state (indicated by the row number). So $\max_a Q(s_i, a)$ represents the maximum state-action value for state $s_i$, which is the estimate of the best action to take in state $s_i$. Thus in the update equation, Equation (5.1) above, at time *t*, an agent in state $s_t \in S$ chooses action $a_t \in A$ and a reward

$r_{t+1}$ is received from the environment at time *t+1*. The update function then updates the state-action value pair $Q(s_s, a_t)$ according to Equation (5.1). The values $\alpha$ and $\gamma$ represents the values of the step-size, and discount factors respectively.

The step-size is a small fraction that influences the rate of learning, and if constant, allows us to effectively track the optimal state-action function Q* as a non-stationary problem. The discount factor determines the present value of future rewards and lies in the range $0<\gamma<1$. If $\gamma$ is set at zero (0), then the agent is only concerned with maximizing the immediate reward, and as it approaches one (1), the agent becomes more prudent and takes into account future rewards. The algorithm implementing the update function in Equation (5.1) is shown in *Chapter 2 Literature Review*.

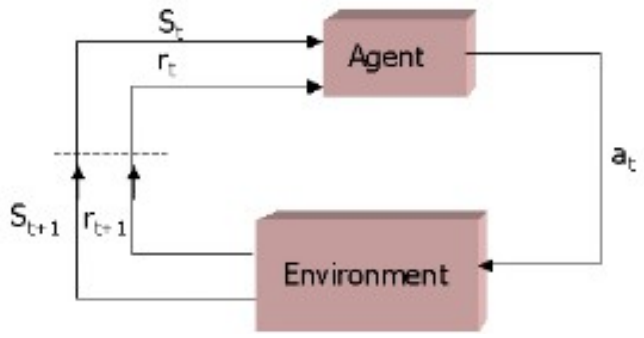


*Figure 5.1: Traditional reinforcement learning update cycle (Sutton and Barto, 1998)*

Coupled with the Q-Learning update function in Equation (5.1) is the representation of the reinforcement learning update cycle shown in Figure 5.1. In Figure 5.1 the agent receives two signals from the environment, the current state of the environment from the agents perspective (agents state), $s_t$, and the current reward (if any), $r_t$ received from the environment for the previous action. These signals are received at time *t* and then first utilized by the agent using Equation (5.1) to update its state-action matrix *Q*[], based on the reward received. The agent then determines an action, $a_t$, to be taken at time *t* based on the current state, which will affect the environment, possibly changing the agents state. In the text time step, the new state and resultant reward for performing the action in the previous time step are passed to the agent for the next iteration of the update cycle. This cycles at each time step while the simulation continues.

It should be noted that in Figure 5.1, the state and reward signals are shown as originating from the environment. In reinforcement learning these signals are considered external from the agent for convenience of explanation. However, in reality, simulation systems are often constructed with the signals generated by some 'sensing' component within the agent itself. This is not inconsistent with the theory as the agents sensing component will perceive the state of the agent and its reward by observing the effects on the environment of the agent's actions. This can be considered external to that component of the agent which utilizes the reinforcement learning functions to update the state-action matrix and to choose an optimal action.

The above discussion describes the operation of a single reinforcement-learning agent learning from interaction with the environment. This research is concerned with a team of such agents that will communicate their successes and failures to each other to possibly enhance the learning and thus success of the group as a whole. Thus a fundamental question at this point becomes, '*How will these agents communicate such knowledge to their team members?*'. Once we determine how the agents will communicate, then we can then develop a formalism as a prelude to devising a modification for Equation (5.1) to take into account multiple updates to the state-action matrix.

In a discrete-time simulation using the traditional model of reinforcement learning and a single agent, the update cycle shown in Figure 5.1 is executed once at each simulated time step. We define a *learning event* as a single update to an agent's action value matrix. Thus in the traditional update cycle the agent will experience at most one learning event per time unit. Given the relatively simple nature of the learning that takes place with the Q-learning update function in Equation (5.1), to achieve cooperation in the learning process we can allow the agents to explicitly share their learning experiences (or learning events). A learning event is communicated in a message from one agent to another in the simulation. In order for an agent to process the learning event contained in a message, it needs to know not only the reward that was received by the other agent from the simulation, but this reward must be placed in context by the corresponding state-action pair that generated it, along with the resultant state. Thus, the structure of the messages required to allow a group of agents to share learning

experiences is a 4-tuple of the form $\langle s, a, r, s' \rangle$. Where $s \in S$ (set of states), $a \in A_s$ (set of actions for state S), *r* is the reward received and $s' \in S$ is the resulting state.

These messages carry enough information about learning events from other group members for the agent to be able to update its own state-action matrix accordingly, via the extra learning events experienced by the other agents. These secondary learning events will be slightly older than an agent's own, but will be only one time step behind. Thus, the age of the extra input signal should not be a significant concern. More formally, we can denote the set of agents in a multi-agent system by *X*, then a multi-agent system with *n* agents is given by

$$X = \{x_1, x_2, \ldots x_n\}$$

Now, if we assume that communication in the simulation is a function of discrete time, we can denote a message between two agents $x_i$ and $x_j$ at time *t* as

$$m_{i,j}(t) : x_i \rightarrow x_j$$

(where each $m_{i,j}(t)$ is of the form $\langle s, a, r, s' \rangle$ as described above). The total communication for *X* at time *t* is then given by

$$M(t) = \{m_{i,j}(t)\}$$

and the messages received by agent $x_j$ at time *t* is denoted by

$$M_j(t) = \{m_{i,j}(t)\}_{i=1..n, i \neq j} \subseteq M(t)$$

## 5.2 Model for Co-operative Learning

There are a number of ways for agents to cooperate in trying to achieve a common goal, and one of these is a form of explicit cooperation, where the agents share their learning events with each other. In the previous section we described the elements (or form) of a message which can be used to share a learning event. We can now consolidate this information to develop a new form of Equation (5.1) to be used in a multi-agent discrete-time simulation utilizing Q-Learning as the agents' underlying learning algorithm. In this type of modified simulation, each agent $x_i \in X$ has at most one learning event per discrete time step. Thus at time *t*, each $x_i \in X$ will receive its reward $r_t$ for the action $a_{t-1}$ that it performed at time *t-1*, resulting in the new state $s_t$ at time *t*. Each of the $x_i \in X$ will then send this learning event in a message of the form $\langle s_{t-1}, a_{t-1}, r_t, s'_t \rangle$ at time *t* to all other agents, that is, a set of messages $\{m_{i,j}(t)\}_{j=1\ldots n, i \neq j}$.

At time *t+1*, agent $x_j$ will update its state-action matrix in the Q-learning algorithm based on the reward received from the environment at time *t+1*. Additionally agent $x_j$ will utilize the messages $M_j(t) = \{m_{i,j}(t)\}_{i=1\ldots n, i \neq j} \subseteq M(t)$ sent by the other *n-1* agents to further incorporate these external learning events into its own state-action matrix. In this way, each $x_j \in X$ will share its fellow agents' learning experiences. The input signal to the agent depicted in Figure 5.1, now becomes more complex, and can be depicted in Figure 5.2

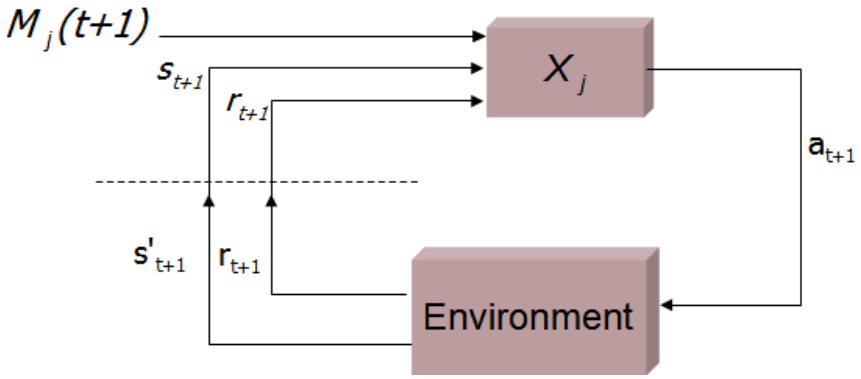


*Figure 5.2: Updated reinforcement learning input signal*

This input signal in the revised reinforcement learning update cycle is now complicated by $M_j(t+1)$. Each of the messages in $M_j(t+1)$ are of the form $\langle s_{t-1}, a_{t-1}, r_t, s'_t \rangle$ since the message was sent at time *t*, generated from actions taken in time *t-1* and received at time *t+1*. Since the agents $X=\{x_1, x_2, \ldots x_n\}$ are homogeneous, the message does not contain the sending agent. This information may be useful to record if trust issues become important and would be a relatively simple matter to update the message to a form $\langle x_i, s_{t-1}, a_{t-1}, r_t, s'_t \rangle$. For simplicity purposes, the sending agent is not utilized and transmitted in this research.

To accommodate the modified input signal shown in Figure 5.2, we need to modify the standard Q-learning update function presented in Equation (5.1). The new model for cooperative learning developed for this research is given in Equation (5.2). This new update function additionally processes all the $\langle s_{t-2}, a_{t-2}, r_{t-1}, s'_{t-1} \rangle$ messages generated for each $m_{i,j}(t) \in M_j(t)$ representing all learning events taking place for $x_j$ at time *t-1*.

$$\left\{ \begin{array}{l} Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \\ \\ \forall\ m_{i,j}(t-1) \in M_j(t): \\ \qquad Q(s_{t-2}, a_{t-2}) \leftarrow Q(s_{t-2}, a_{t-2}) + \\ \qquad\quad \alpha \left[ r_{t-1} + \gamma \max_a Q(s'_{t-1}, a_{t-1}) - Q(s_{t-2}, a_{t-2}) \right] \end{array} \right\} \qquad (5.2)$$

Equation (5.2) processes the agent's own reward received at time *t* for the action it chose to perform at time *t-1,* as well as essentially applying Equation (5.1) for each $\langle s_{t-2}, a_{t-2}, r_{t-1}, s'_{t-1} \rangle$ message in $M_j(t)$.

There are a couple of points to note about Equation (5.2). First, It can be noted from the messages in $M_j(t)$, that the actions that generated the rewards occurred two

discrete time steps prior to the time step in which the corresponding learning event is being used to update the agents state-action matrix. Thus the learning events shared by the team of agents are older learning events, but only by two discrete time steps. It is not anticipated that the age of the learning events (given that it is only two time-steps behind) would be any significant problem. An older action may have resulted in a new state that is no longer relevant in the current time step, nonetheless, the action was valid and generated a reward that the agent can still assimilate into its state-action matrix. Second, as the actions by all the agents are randomized before being applied, (see Previous Chapter), it is impossible to receive the messages from all other agents in the order that the actions were applied and hence result in the rewards that were given. Thus it is unknown whether processing the learning events in the actual order they occurred would make any difference. However, it should be noted that if we considered a real scenario in the underlying domain involving human agents, it would also be highly unlikely that incoming messages would be received in the exact order that the learning events took place. Thus the simulation utilizing Equation (5.2) is in all likelihood closer to reality.

By applying each of the learning events contained in $M_j(t)$ to its own state-action matrix through the update function in Equation (5.2), each $x_j$ is incorporating the learning events of its team members. Thus at each time-step, an agent can potentially increase its learning rate based on the team learning.

The learning rate $l(x_j)$ of an agent $x_j$ can be determined by the number of learning events *e*, that agent $x_j$ experiences per discrete time unit *t*. The learning rate for agent $x_j$ is then given by the function

$$l(x_j) = \frac{e}{t}$$

For the group of agents $X = \{x_1, x_2, \ldots x_n\}$ described above, the learning rate then becomes:

$$l(X) = \sum_{j=1\ldots n} l(x_j)$$

In the Q-learning algorithm described in Equation (5.1), each agent receives a single reward from the environment at time *t+1* after all actions have been processed at time *t*. This reduces the learning rate per agent to a maximum of one learning event per time unit. So the learning rate for an agent $x_j$ is then bounded by the simple function

$$l(x_j) = 1$$

Thus, for the group of Q-learning agents *X* in the simulation, the learning rate of the group is bounded by the linear function:

$$l(X) = \sum_{j=1\ldots n} 1 = n$$

The modified Q-learning algorithm described in Equation (5.2) lets the individual agents share their learning experiences with the rest of the group, hence increasing the learning rate. If each of the *n* agents experiences a single learning event at time *t (the maximum),* this can be passed on to each of the other *n-1* agents in the group at time *t+1*. The maximum learning rate for the group at time *t+1* would then occur when each of $x_j$ agent's experiences another learning event, as well as receiving and processing information from the other *n-1* agents contained in $M_j(t+1)$. In such a case, the group learning rate is bounded by the function:

$$l(X) = n + n\ (\ n - 1\ ) = n^2$$

This maximum learning rate assumes that the communication of the Q-learning agents is global and not restricted by range. If range restrictions apply, then the above maximum rate is conditional upon the proximity of agents to each other during the simulation, and would be reduced accordingly.

### 5.2.1 Task Performance Gains

We know from the literature that allowing agents in a multi-agent system to communicate during the learning process will increase the task performance of the group as a whole. Additionally we know from many of the research papers that communication in a multi-agent systems helps accelerate the learning process, for example (Darbyshire and Wang, 2003, Kelly and Keating, 1998). Thus we know the cause-effect relationship, but what still seems unsatisfactorily documented in the literature is a definitive answer of where the changes due to communication specifically effect either the agents or the agent system as a whole. Does the communication affect the metrics of the agent system which accounts for the increased performance, or does this actually have an effect on the agents themselves, or both? The exact mechanics of the communication effects requires investigation. A detailed knowledge of these effects on the agents themselves in such an environment, might help to provide avenues for further exploration of utilizing communication for group learning. These are some of the questions we investigate in the following section.

## 5.3 Simulation Description

The architecture of the simulation is basically a non-supervised multi-agent system representing a military distillation with two opposing teams who combat for survival. This simulation is designed to study combat at the conceptual level. The simulation world is represented by *W*, a continuous world where agent movement is determined by a vector consisting of angle and distance (based on a speed variable). The simulation is modeling a specific problem and utilizes a finite state and action space. Time is discrete, and for each time step $t \in \mathbb{N}$, each agent observes the world, determines its current state, and chooses an appropriate action to perform. All actions are performed at the end of each time step, with the sequenced randomized before execution to prevent bias.

There are two groups of agents operating within *W*. *Y* and X. $Y = \{y_1, y_2, \ldots y_n\}$ represents the control group (*Red Team*), with the behavior of each $y_i \in Y$ governed by a rigid control paradigm consisting of a series of weighted

values for various behavioral aspects. That is, each of the $y_i$ will have defined a number of behavior variables that determine how it will react in any given situation: willingness to engage; willingness to explore; tendency to obey orders etc. Each of these behavior variables is pre-assigned a weighted value, usually between zero and one. These weights are used by the governing algorithm to determine each agent's behavior at each simulation time step. This is a typical agent control paradigm prevalent in such distillations, where the weights can be modified to study the outcomes given different types of expected behavior in the agents. Agents within this group represent a highly trained team, but are not the focus of this research and are thus considered part of the environment for the remaining agents.

The group of agents $X = \{x_1, x_2, \ldots x_n\}$ is the focus of this research (*Blue Team*), and each $x_i \in X$ is controlled by the modified reinforcement learning algorithm given in Equation (5.2).

### 5.3.1 Initial State Space

The choice of state space to facilitate the reinforcement learning is always difficult, and the states and actions will vary greatly from application to application. The construction of an adequate state space for a particular reinforcement-learning problem is not adequately dealt with in the literature. Indeed, it has been mentioned that construction of a good state space is more art than science (Sutton and Barto, 1998). However, the states are the basis of making choices for which actions to choose and how the state-space is partitioned can greatly affect the performance. The states should be chosen to give the agent enough information from the environment to make reasonable choices in order to maximize the reward. Thus, the choice of states should reflect what is of 'interest' to the agent for pursuing the maximization of the reward function.

The agent environment detailed previously is continuous and doesn't rely on a grid representation for the world, but rather a coordinate system with vectors for representing direction and speed. Thus, the initial state space for the Q-learning agents constructed quantizes and abstracts the points of interest for these agents, and is

represented by four integer variables used to determine the possible state of each agent. These variables and their possible values are shown in Table 5.1. The finite state space *S* is determined by a number of discrete variables and their possible range of values. Each of the variables targets an attribute that will be of interest to the agent as it pursues its goal. Each agent can be in one of 36 possible states based on the possible permutations of the state space variables given the values shown in Table 5.1.

*Table 5.1: Initial state space variables*

| attribute description | possible values |
|---|---|
| agents health | 0 – bad 1 - good |
| Average distance to enemy | 0 – near, 1 - far |
| ratio of enemy to friendly in visual range<br>ratio of enemy to friendly in firing range | 0 – more friendly<br>1 – equal<br>2 – more enemy |

The justification for the initial state space lies in the fact that the state space must allow the agents to discern what may be interesting to them as they pursue their reward. The reward function (discussed in Section *5.3.2 Reward Function*) provides stimulus for the agents behavior as they receive rewards through their actions, but the state space provides them with information that helps them to choose actions to maximize the reward. Since the reward function for these agents is based simply on positive rewards for reducing the health of enemy agents, and negative rewards for having their own health points reduced, the state space must be geared to helping agents discern what might be interesting in achieving positive rewards. An important piece of information to an agent is their own health state. If it is high, more risky actions may achieve higher positive rewards, yet if it is low, more prudent actions might yield better results in the long run. The distance to the enemy would be of interest to an agent, as the agent can

then elect to move closer to possibly pursue a positive reward in future, or increase this distance if the agents' health is poor. Additionally the ratio of enemy agents to friendly agents in the agents' immediate area could be useful in leading to positive rewards. If the value is 0, then an agent can move closer to the enemy with a greater chance of remaining alive while possibly gaining a positive reward. If the value is 2, then there is more likelihood of negative rewards if moving closer.

Each of the agents in both teams X and Y, were initialized with 100 health points, and health points were deducted for a perceived wounding, or set to zero for a kill. The agents health state space variable was partitioned into two intervals and allocated the following values:

$$agents\ health = \begin{Bmatrix} 0 & health & < & 50 \\ 1 & health & \geq & 50 \end{Bmatrix}$$

The average distance of the agent to all visible enemy agents is calculated and partitioned into two intervals and allocated the following values:

$$enemy\ average\ distance = \begin{Bmatrix} 0 & avg\ dist & \leq & firing\ range\ distance \\ 1 & avg\ dist & > & firing\ range\ distance \end{Bmatrix}$$

The next two state space variables are based on ratios between the perceived enemy agents and the perceived team members within certain ranges and each can have three possible values. These are set as follows:

$$visual\ range\ ratio = \begin{Bmatrix} 0 & more\ team\ members \\ 1 & equal\ amounts \\ 2 & more\ enemies \end{Bmatrix}$$

$$firing\ range\ ratio = \begin{Bmatrix} 0 & more\ team\ members \\ 1 & equal\ amounts \\ 2 & more\ enemies \end{Bmatrix}$$

Thus each state is represented by a 4-tuple of the form

< agentHealth, avgEnemyDist, visualRangeRatio, firingRangeRatio>

There is one assumption with the state space variables, and this is that the firing range is less than the visual range. However this does not preclude the existence of any possible permutations of the state space variables. It is possible for an agent to be in any one of the 36 possible states.

### 5.3.2 Reward Function

The reward function develops the agent behavior, but is dependent on the state space and available actions. The state space must allow the agent to discern or partition itself into states that are interesting in the pursuit of its reward. The available actions must allow the agent the possibility of attaining its reward. Given the combative nature of the simulation, the objective of each team is to engage in combat with the opposing team until one team is eliminated. The reward function for the Q-learning team must be designed to elicit this behavior. The simple reward structure used in this simulation to develop this behavior is illustrated in Table 5.2.

*Table 5.2: Reward function for the simulation*

| | **reward value** | **action** |
|---|---|---|
| $r_t\ (s_i,a_i) =$ | 0.5 | damage to enemy |
| | -0.5 | receives damage |
| | 1.0 | kills enemy |
| | 0 | otherwise |

While the reward does ultimately influence what we want the agents to learn, making it too complex could elicit behavior in the agents that otherwise might not arise as an emergent property of the simulation. Thus the structure of the reward function was kept at an extremely simple level.

### 5.3.3 Initial Action List

The actions represent the available choices which can be made by the agent at each time-step, and should enable the agent to perform some task based around receiving some reward from the reward function discussed in the previous section.. For each state, an agent has a choice of six possible actions. A initial set of actions that were made available to the Q-learning agents is shown in

*Table 5.3: Initial set of possible actions*

| **Possible Actions** |
|---|
| Do nothing |
| Move away from enemy locus |
| Move to closest enemy agent |
| Fire at the closest enemy agent |
| Move to and fire at closest enemy agent |
| explore |

Not all actions are relevant for all states. Even though the choice of initial states discussed above uses ratios of enemy to friendly agents and not specific numbers, there will be times when there are no enemy agents in visual range. Thus the actions involving moving to or from the enemy and firing at the enemy would not be applicable in these circumstances. This does in itself suggest that a refinement of the actions may be needed, however some initial trials were undertaken with these actions.

At each time step, the Q-learning algorithm will choose an action to perform from the list of possible actions for its current state. The action is chosen using a simple ε-greedy action selection. That is, the action will be chosen to maximize the expected reward, but will randomly choose an action based on a probability factor of ε. Thus, we can try to balance reward maximization with exploration by choosing other actions. This factor was set to 0.1 for the initial trials.

## 5.4 Experimental Design

While the scope of this research is to study the effects of communication on learning rates, this section describes the framework of the experiments conducted, and the design of the Q-learning agents without communication abilities. Before a study can be conducted on the effects of communication between Q-learning agents, a benchmark needs to be established with non-communicating Q-learning agents, against which we can then measure the effects of communication. Following the initial benchmark trial, further simulation 'runs' were conducted, each designed to test either full communication of all learning events, or partial communication of different types of learning events.. Each of the simulation trials consists of 2000 simulation runs of the distillation simulation software described in the previous chapter. Two thousand (2000) simulation runs was deemed enough in which to discern dominant trends for each of the trials conducted.

Each simulation run consisted of two mutually hostile teams of agents, 'Red' and 'Blue' teams respectively, and each team contains 15 agents. Red team agents are controlled by the standard instinctual agent control paradigm as previously discussed, and the Blue team agents implement the Q-Learning algorithm from Equation (5.1) as described in previous sections. Both teams were given the same parameters: 100 health points per agent (1500 per team); visual capabilities; movement capabilities; and weapon capabilities. Each agent carries the equivalent of a single projectile weapon capable of firing one directed rounded per simulated time unit. Each of the teams are initialized at opposite diagonal corners of a 500 x 500 2-dimensional world, with the Q-learning agents in the lower right corner and initially set in the direction as shown in Figure 5.3. The world is not represented as a discrete grid but is represented as a continuous 2-D space.

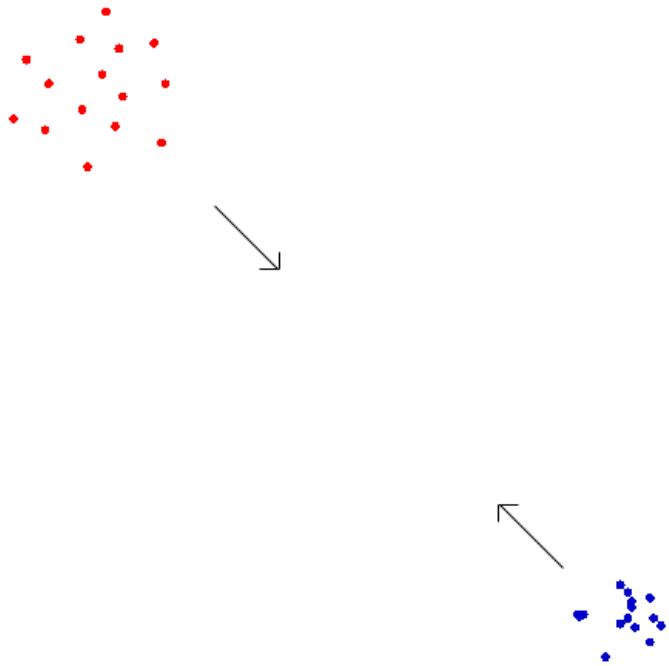

*Figure 5.3: Agent team initialization*

Both of the agent teams were initially set in the directions as shown in Figure 5.3. This does not bias the simulation at all due to the structure of the reward function. For the Q-learning team (b*lue team*), no learning events are generated until the agents come into visual range of each other. Once this occurs, the reinforcement learning algorithm then takes over and the Q-learning team begins to learn as learning events are generated. The Instinctual team (*red team*) agents will then begin to be governed by their built-in behavior patterns. The agents have limited vision and can only observe a portion of the world at any one time, thus we have the equivalent of a Partially Observable Markov Decision Process.

The simulation itself uses a discrete time-step algorithm, and at each time step, each agent, regardless of the team, enters a 'thinking cycle', where it perceives the world and decides on a course of action. Once the action is decided, the agent stores its proposed action in a system queue. In this way, each of the agents can decide their course of action based on the current 'snapshot' of the world at the current time-step.

Once all the actions have been recorded, the simulation processes them, before starting over again with a new 'thinking-cycle' at the next time-step. Each of the 2000 runs in an experimental trial executes for at most 1000 time-steps. This is long enough for the final outcome of the simulation run to be decided. Each of the 2000 runs was then averaged to produce the averages for the specified trial.

The two constants $\alpha$ and $\gamma$ in both Equation (5.1) and Equation (5.2) above represent the step-size and the discount factor respectively. The step-size is a small fraction that influences the rate of learning, and if constant, allows us to effectively track the optimal state-action function Q* as a non-stationary problem. The discount factor determines the present value of future rewards and lies in the range $0 < \gamma < 1$. If $\gamma$ is set at 0, then the agent is only concerned with maximizing the immediate reward, and as it approaches 1, the agent becomes more prudent and takes into account future rewards. For all trials utilizing both Equation (5.1) and Equation (5.2), the values of the constants used in the simulations are shown in Table 5.4. Minor variations to these values made no significant difference to the outcomes of the trials.

*Table 5.4 List of parameters and their values for Simulation trials*

| **parameter** | **description** | **value** |
|---|---|---|
| $\alpha$ | Step-size | 0.1 |
| $\gamma$ | Discount factor | 0.1 |
| $\epsilon$ | E-greedy randomness probability | 0.08 |

The constant $\epsilon$ shown in Table 5.4 is use in an $\epsilon$-greedy policy and represents the probability that a random action will be chosen instead of one that maximizes the value of the reward. In this case the value of $\epsilon$ is set at 0.8. It has to be large enough to be useful in allowing untried actions a chance at being tried, yet in the limited time-steps the simulation has to maximize the reward it cannot be too large.

## 5.5 Initial Benchmark Results

The initial simulation trial was used to establish a benchmark for the latter trials. The purpose of the benchmark trial was to utilize the original Q-learning function shown in Equation (5.1) and thus establish the benchmark for the Q-learning team. This provides a measure for which all trials utilizing communication, and thus the modified Q-learning function in Equation (5.2) can then be compared. The initial question was now, '*how well would the Q-Learning agents do in the simulated combat*' against a similar team of instinctual agents, representing a highly trained team.

The simple goal of the exercise was to set the two teams against each other to gauge the success of the reinforcement learning in this scenario. This will serve two purposes, to give an idea of the possible success of the Q-Learning team, and to provide a benchmark. One way to measure the success or effectiveness of the Q-learning algorithm in this environment is to compare the relative damage caused by each team against the other over time. Figure 5.4 shows a plot of the damage caused over time by each of the two teams. This plot represents the average taken over all runs in the trial, as previously indicated. The damage (vertical axis), is represented by the reducing health of the two teams over time as the combat progresses. Each team member begins with 100 health points, or a total of 1500 per team. The lower, blue plot represents the Q-Learning agent team (*blue team*), while the red (upper) plot represents the instinctual agents team (*red team*). The standard deviation value $\sigma$ for the average blue plot line in Figure 5.4 is 11.02.

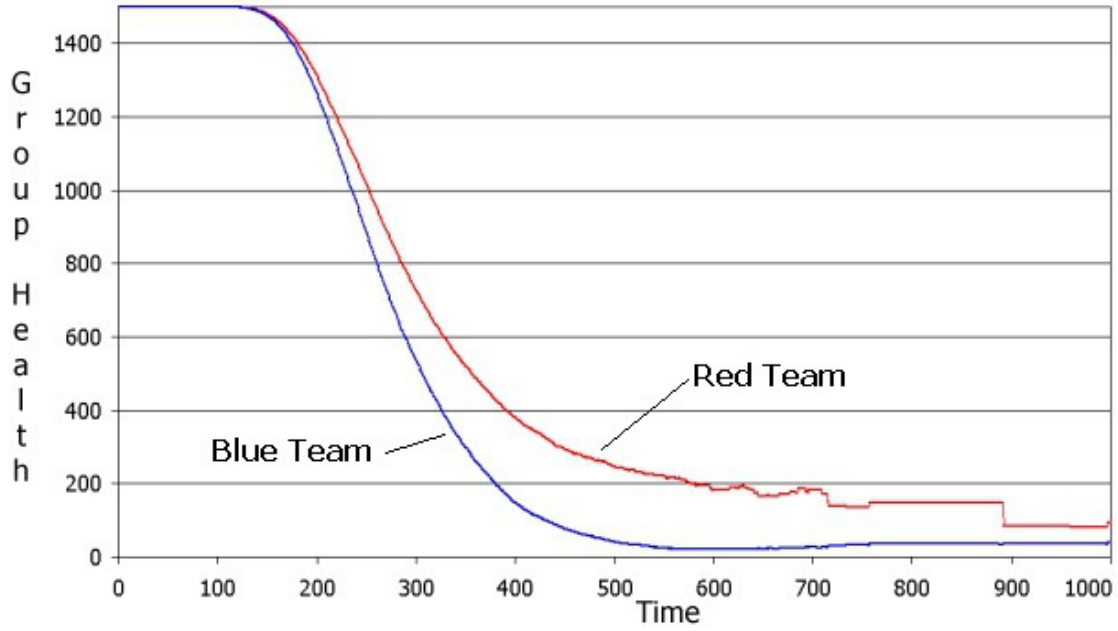


*Figure 5.4: Plot of damage over time for initial benchmark trial*

As can be seen from Figure 5.4, the Q-learning agent team does significantly worse, than the Instinctual agents, winning a decisive victory in only 3.2% of encounters (runs for the trial). A decisive victory is classed as an encounter when all members of one team are killed. While this result was expected, a more detailed analysis can provide some insights to help in addressing possible areas of enhancement to the algorithm. Also, learning did not really improve much beyond the 400 simulated time step mark. These results tend to back up similar findings in (Gugel et al., 2001).

As the benchmark trial consisted of 1000 simulated time-steps in duration, each of the Q-Learning agents has a potential to perform 1000 iterations of the learning cycle shown in Figure 5.1. However, as shown in Figure 5.4, on average, the initial contact between the two groups (in terms of first damage inflicted) takes place at approx time-step 130, and lasts for approximately another 300 time-steps. As the reward structure only allocates rewards based on damage inflicted or received, no learning takes place when the two groups are not in conflict. So while the trials were 1000 time-steps in duration, effectively all the learning by the agents must take place during a short burst of activity. A more holistic understanding can be gained by a detailed analysis of some of the individual trials.

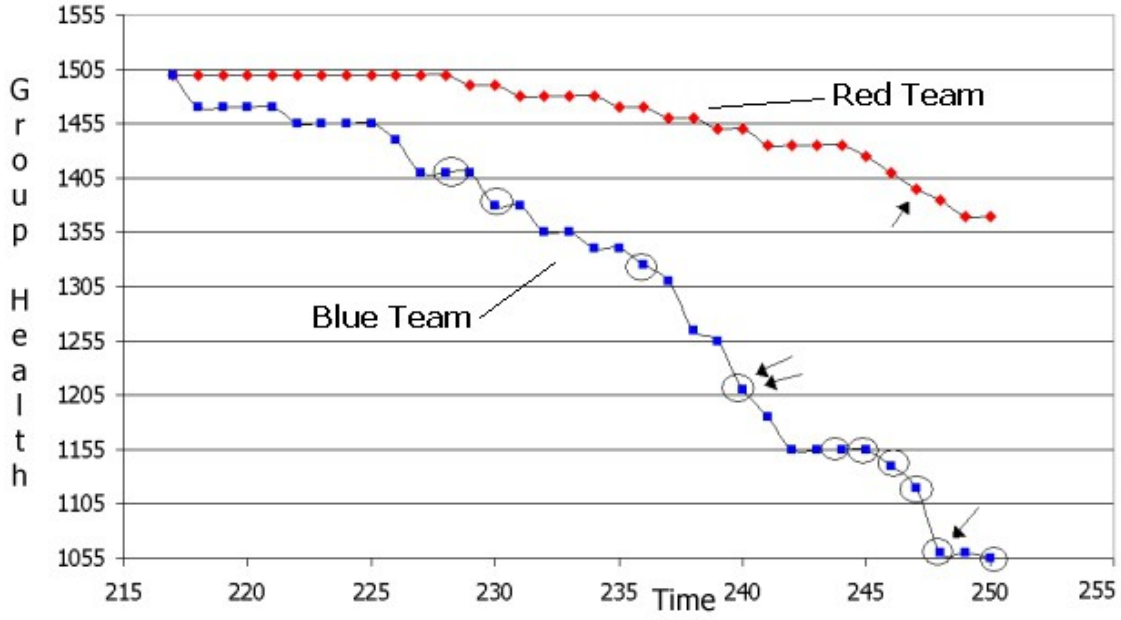


*Figure 5.5: Initial benchmark trial individual simulation run analysis*

A plot of the activity in a typical simulation run for the first 35 simulation time-steps after initial contact is shown in Figure 5.5. Again, the horizontal axis represents the simulation time, while the vertical axis represents decreasing team health. The lower blue plot-line represents the Q-learning team, and the upper red plot-line represents the red team. This chart only displays a portion of the trials activity for brevity. As indicated, after first contact at time 217, the first successful action of a Q-Learning agent doesn't occur till time 228, indicated by the circled plot in Figure 5.5 at this time. This would represent a positive outcome for the agent involved, and reinforce the appropriate action-state pair. The circled points on the Q-Learning agent graph represent points in time that different agents receive a positive reinforcement from the simulation.

The directed arrows on both plot-lines in Figure 5.5 represent points in time when the opposing team has killed an agent on the team indicated. Thus, in Figure 5.5, at time 250, three blue agents have been killed to one red agent. As the time progresses, more Q-Learning agents receive positive rewards, with 6 different agents learning successful action-state pairs between times 244 and 250. But as the simulation trial progresses, the learning agents are continually taking damage as they progress through

the learning iterations, which would tend to negatively re-enforce some of the actions they have just come to learn. So while the agents are learning, the rate of learning is not fast enough to compensate for the negative reinforcement, thus the outcome indicated in Figure 5.4.

Inspection of the state-action matrices for the Q-Learning agents killed early confirms this. The state-action matrix for all 3 learning agents killed in Figure 5.5 show all values (except one) either zero or negative. Typically for all agents killed, the state-action matrices only show activity in very few of the 36 possible states over all the simulation runs. This indicates that perhaps the state space variables used to determine possible states needed refinement (or expansion). Before testing the communication to increase learning rates, a refinement of the state space variables and possible actions were undertaken..

## 5.6 State Space Variables and Actions Refinement

The original state space variables and possible agent actions were revisited with a view to further partitioning the state space into a more refined group of states. Along with an expanded set of actions, this could possibly give the agent more chance of finding successful actions resulting in transitions to alternative states, and possibly positive rewards. The first set of states only allowed the agent to partition the state space based on the ratios of enemy agents to friendly agents. In cases where no enemy agents were in visual range, these ratios became simplistic and defaulted to one (1). It would perhaps be more prudent to allow the agents to divide the state space further based on whether there were enemy agents in either visual range or firing range, as well as discern the ratios. Additionally, the concept of a wounded enemy agent could be interesting to Q-learning agent, The reward function allocates a 0.5 reward for a wounding of an enemy agent, but a 1.0 reward for the killing of an enemy agent. It is more likely that firing on a wounded agent would result in a kill rather than firing on a non-wounded agent.

The state space variables were modified to include the seven state variables as shown in Table 5.5.

*Table 5.5: Modified list of state space variables*

| **attribute description** | **possible values** |
|---|---|
| enemy in visual range<br>enemy wounded in visual range<br>enemy in firing range<br>wounded enemy in firing range | 0 – no 1 - yes |
| agents health | 0 – bad 1 - good |
| ratio of enemy to friendly in visual range<br>ratio of enemy to friendly in firing range | 0 – more friendly<br>1 – equal<br>2 – more enemy |

The finite state space $S$ is now determined by the seven (7) discrete variables shown in Table 5.5. The first four state space variables are simple Boolean functions that assign the value of zero (0) or one (1) based on whether enemy agents are in either visual or firing range, eg:

$$enemy\ agent\ visible = \begin{Bmatrix} 0 & no \\ 1 & yes \end{Bmatrix}$$

$$wounded\ enemy\ agent\ visible = \begin{Bmatrix} 0 & no \\ 1 & yes \end{Bmatrix}$$

$$enemy\ agent\ within\ \ firing\ range = \begin{Bmatrix} 0 & no \\ 1 & yes \end{Bmatrix}$$

$$wounded\ enemy\ agent\ within\ \ firing\ range = \begin{Bmatrix} 0 & no \\ 1 & yes \end{Bmatrix}$$

The agents health state space variable is partitioned as before, with two possible values depending on the remaining health points.

$$agents\ health = \begin{Bmatrix} 0 & health & < & 50 \\ 1 & health & \geq & 50 \end{Bmatrix}$$

The two state variables dealing with the ratios of enemy to friendly agents in either visual or firing range have one of three possible values as before

$$visual\ range\ ratio = \begin{Bmatrix} 0 & more\ team\ members \\ 1 & equal\ amounts \\ 2 & more\ enemies \end{Bmatrix}$$

$$firing\ range\ ratio = \begin{Bmatrix} 0 & more\ team\ members \\ 1 & equal\ amounts \\ 2 & more\ enemies \end{Bmatrix}$$

Each state can now be represented by a 7-tuple of the form

< (EiVR, WiVR, EiFR, WiFR, heath, VisualEFRatio, firingEFRatio)>

where

EiVR Enemy in Visual Range

WiVR Wounded in Visual Range

EiFR Enemy in Firing Range

WiFR Wounded in Firing Range

health this is the quantized health state

visualEFRatio quantized value for quantity e/f in visual range

firingEFRatio quantized value for quantity e/f in visual range

The total number of permutations of the values of the state variables, given the possible values above is now 288. Each state is represented by a 7-tuple as above, such

as <1,0,1,0,0,0,2> with each number being one of the possible values for the different attributes represented in the tuple. However, again, we retain the assumption that the firing range is less than the visual range, thus not all states are possible as they are not orthogonal. If the visual range is greater than the firing range, then if we have no enemy in visual range we cannot have any in firing range, and the two ratio attributes are meaningless. Eliminating impossible states and inconsistencies leaves a possible 68 states. These can be seen in *Appendix B.1*.

The reward function was retained as described in Section 5.3.2 Reward Function.

### 5.6.1 Action Refinements

The number of actions were increased from the original six (6) to 11, with specific actions to allow possible targeting of wounded enemy. These actions were designed to be used in conjunction with the expanded state space variables. The list of actions can be seen in Table 5.6

*Table 5.6: Updated set of possible agent actions*

| Possible Actions |
|---|
| Do nothing |
| Explore |
| Move away from enemy locus |
| Move to closest enemy agent |
| Move to closest wounded enemy |
| Fire at the closest enemy agent |
| Fire at closest wounded enemy |
| Move to and fire at closest enemy agent |
| Move to and fire at the closest wounded enemy |
| move away from enemy and fire at the closest |
| move away from enemy and fire at the closest wounded |

Again, not all actions are applicable to all states. For instance, if there is no enemy or wounded enemy in either visual or firing range, then all actions involving firing at the enemy are not possible. Likewise, if there is no enemy in visual range, then actions involving to or from an enemy agent are also not possible. The complete list of possible states, and the actions applicable to each state are shown in Appendix 1.

Although not all actions are applicable to all states, the implementation of the state-action matrix assumes a homogenous set of actions for each state in its structure. However, when the simulation is initialized a list of all possible actions for each state is loaded ensuring that an inappropriate action is not selected when deciding on which action to take for a given state. The ***State Action Map*** is listed in *Appendix B.1*. This is a standard text file containing a list of all possible actions (see Table 5.6) in abbreviated format, shown on the next line.

```
nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
```

Below the list of all possible actions is a list of all valid states and each valid state contains a list of the possible actions allowed in that state. An excerpt of that section of the file is shown below in Listing 5.1.

```
                    :
(1 0 0 0  1 0 0) nothing explore movaway movto
(1 0 0 0  1 1 0) nothing explore movaway movto
(1 0 0 0  1 2 0) nothing explore movaway movto
(1 0 1 0  0 0 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  0 0 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  0 0 2) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  0 1 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  0 1 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  0 1 2) nothing explore movaway movto fire movtoF movawayF
                    :
```

*Listing 5.1: Excerpt of State Action Map text file*

In Listing 5.1, note that each state is represented as a tuple with the values relating to the set of state variables discussed in section *5.6 State Space Variables and Actions Refinement*, and shown in *Appendix B.1*. For example, in Listing 5.1, the state tuple

(1 0 1 0 0 1 2)

refers to the state in which: enemy agents are in visual range; no wounded enemy agents are in visual range; enemy agents are in firing range; no wounded enemy agents are in firing range; the health points of the current agent are bad (less than 50%); there are equal amounts of enemy and friendly agents in visual range; there are more enemy than friendly agents in firing range. The list of possible actions for this combination of state variables is listed to the right of the state tuple. The *State Action Map* text file is read and interpreted by the simulation engine before the simulation begins execution. It is used to build up the *Q*[] matrix (state-action matrix of the reinforcement learning agents, blue team) as shown in Figure 4.11 and discussed in *Section 4.5 The Thinking Cycle*. The list of possible actions is used to populate the *Actions[]* vector (part of the *Q*[] matrix) and is used by the agents during the *thinking cycle* to decide on a course of action, depending on the values in the corresponding *Values*[] vector of the *Q*[] matrix.

## 5.7 Experimental trials

Given the refinement of the state space variables and the updated action list, the proposed simulation runs were again undertaken. All simulation runs were divided into four (4) comparative simulation trials to to measure the effects of allowing the agents to communicate. These simulation trials are listed below in Table 5.7.

*Table 5.7: Brief description of the four comparative simulation trials to measure the effects of communication on group learning and success*

| Simulation-trial | Description |
|---|---|
| *benchmark* | agents do not communicate and are individual learners |
| *comms1* | agents communicate positive reward learning events only |
| *comms2* | agents communicate negative reward learning events only |
| *comms3* | agents communicate all learning events |

Each simulation trial consisted of 2000 simulation runs using the same parameters and communication methods. The results of the 2000 simulations runs for each trial were then averaged to produce an overall picture of the outcomes of each of the trials.

The *benchmark* trial was undertaken utilizing the standard Q-learning update function in Equation (5.1). This then gives us the benchmark by which to compare the other trials which allow communication between the agents. The other three trials all utilize the updated Q-learning function developed in Equation (5.2). That is, each agent, on experiencing a learning event, will send a message to to all other agents in the Q-learning team, in the form of the tuple detailed in *Section 5.2 Model for Co-operative Learning*. In the next time step in the simulation, each agent $x_j$ receiving such a message will use the information in the message to further update its state-action matrix according to Equation (5.2), using all the messages in $M_j(t)$.

In the comms1 trial, the agents only communicated their learning events associated with positive rewards, while in the comms2 trial, the agents would only communicate their learning events associated with negative rewards. In comms3, the agents communicated all learning events. This separation was undertaken to see if a faster learning rate could be achieved by communicating only one type of learning event. No prior bias was expected in the results, which are detailed in the following sections.

### 5.7.1 Benchmark Trial

In contrast to the original *benchmark* trial undertaken, this benchmark trial utilized the new state space variables and actions refined in section 5.6 State Space Variables and Actions Refinement. Again, the results of the 2000 simulation runs were averaged to produce a damage vs time plot for the new *benchmark* trial. This plot gives us an overview of how the reinforcement learning agents in the benchmark trial fare against their counterparts in terms of the team health.

The results of an experimental run using the new state-space variables and actions for the Q-learning team are shown in Figure 5.6. When comparing Figure 5.4 to Figure 5.6 there are some evident differences. The Q-learning agents fare much better on average when looking at the total damage inflicted, and it takes slightly longer for the two graphs to deviate significantly, indicating that the Q-learning agents fare better earlier on, and throughout the simulation. Also significant is the increase in the number of decisive wins from 3.2% to 24.2%. However, interestingly the point of contact between the two groups comes earlier, and the time of contact where most of the activity occurs is considerably shorter. Thus, these Q-learning agents have less thinking-learning cycles in which to learn successful actions. The standard deviation value $\sigma$ for the average blue plot line in Figure 5.6 is 8.01. This represents a reduction from the $\sigma$ value for the original benchmark trial.

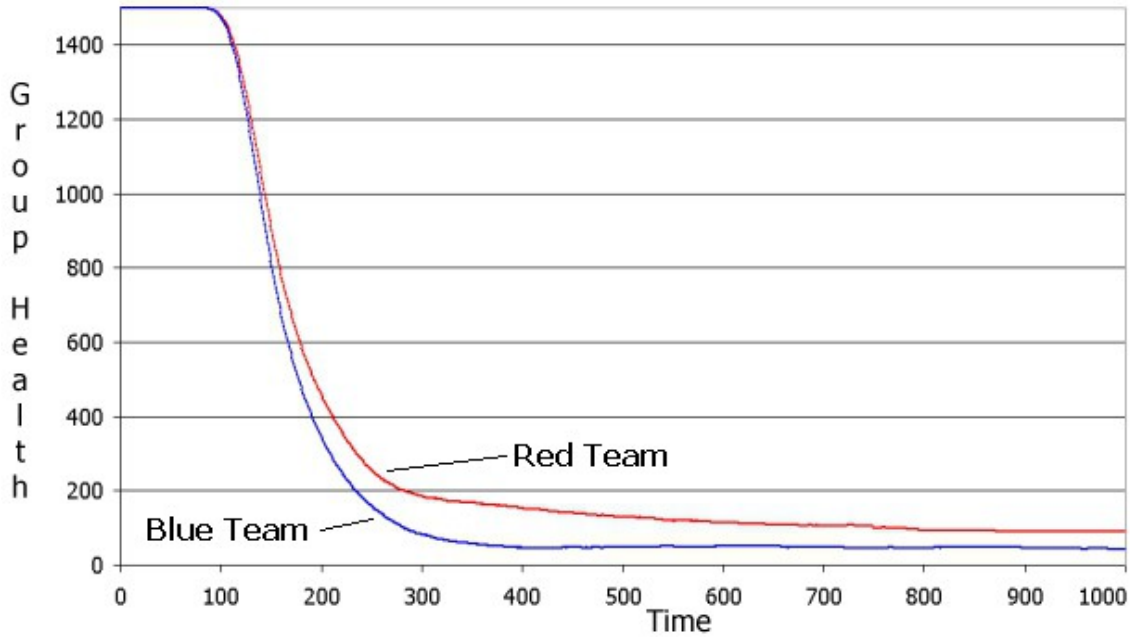


*Figure 5.6: Damage vs time plot for updated benchmark trial*

While this demonstrates the important role of a viable set of actions to the effectiveness of the learning algorithm, the refined state-space variables and possible actions, while improving performance, is still not sufficient to assure the agents the adaptability they need to survive. While each trial runs for 1000 time steps, the hostile nature of the simulation effectively confines the requisite learning to a small time window approximately 200 steps in duration (Figure 5.6). The learning which takes

place after this window of opportunity is not enough to ensure the survival of the team. In fact, the Q-learning algorithm applied in this manner is demonstrative only of individual learning, and not of team learning. The individual agents do learn, but no knowledge is shared to enhance the effectiveness of the team as a whole. We can now increase the learning rate of the team as a whole by allowing the individual agents to share their knowledge.

Throughout the remainder of this chapter, we use the Q-learning algorithm with this set of expanded state-space variables and action set as the benchmark against which we measure the effects of communication.

### 5.7.2 Comms1 Trial

In this trial of 2000 simulation runs, the Q-learning agents were only allowed to communicate the learning events associated with positive rewards. The assumption made here was that perhaps by only communicating the positive rewards, these would not be counter-acted by the negative rewards, thus only positively reinforcing the actions which led to successful outcomes. This might further accelerate the learning process. Again, this trial consisted of 2000 simulation runs averaged to produce a damage vs time plot as shown in Figure 5.7. The standard deviation value $\sigma$ for the average blue plot line in Figure 5.7 is 9.05.

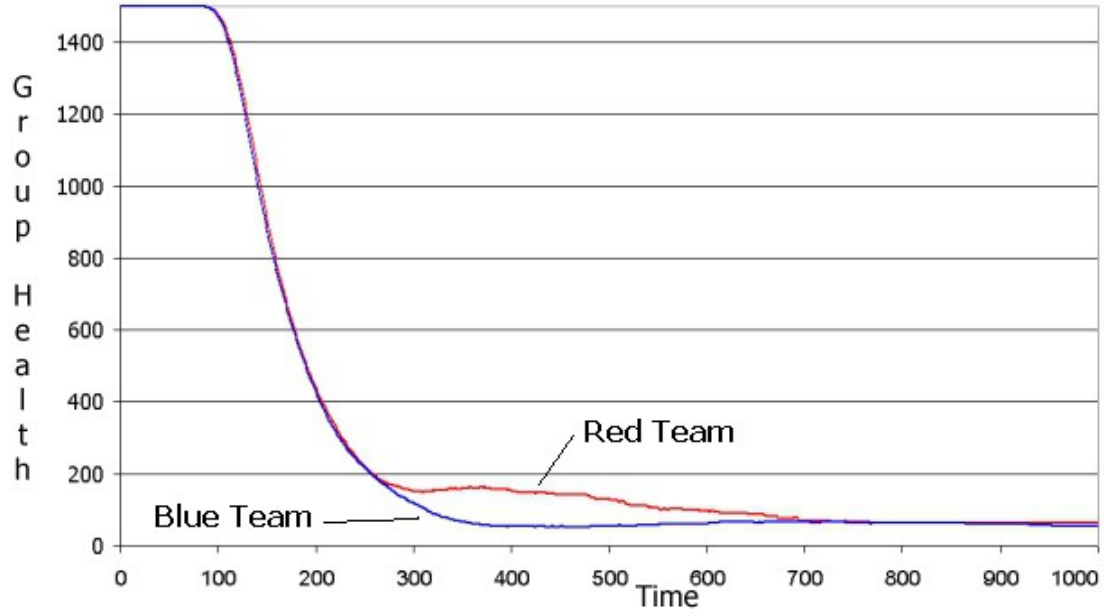


*Figure 5.7: Damage vs time plot for comms1 trial*

Figure 5.7 shows the damage vs. time plot for simulation trial *comms1*. As can be seen in comparison with Figure 5.6, the Q-learning team does significantly better than the *benchmark* with both plots almost overlaying each other initially. The Q-Learning plot represented by the blue (lower plot-line). There is an anomalous section between time steps 270 – 700, which is interesting as the average combat time for decisive victories is approximately 332 time-steps, a short way into this area. The number of decisive victories for the Q-Learning has also significantly increased from 24 percent (Figure 5.10), to 48 percent, while the decisive victories for the instinctual agents decreased from 68 percent to 47 percent.

### 5.7.3 Comms2 Trial

This trial was conducted as a counterpoint to the *comms1* trial. In this trial, the Q-learning agents were only allowed to communicate the learning events associated with negative rewards. While this would not result in successful actions being positively rewarded as in comms1, it would result in unsuccessful actions being negatively rewarded, which may lead to a similar result in that these unsuccessful actions may not be chosen again. Figure 5.8 shows the damage vs time plot for *comms2*. The standard deviation value $\sigma$ for the average blue plot line in Figure 5.8 is 8.30.

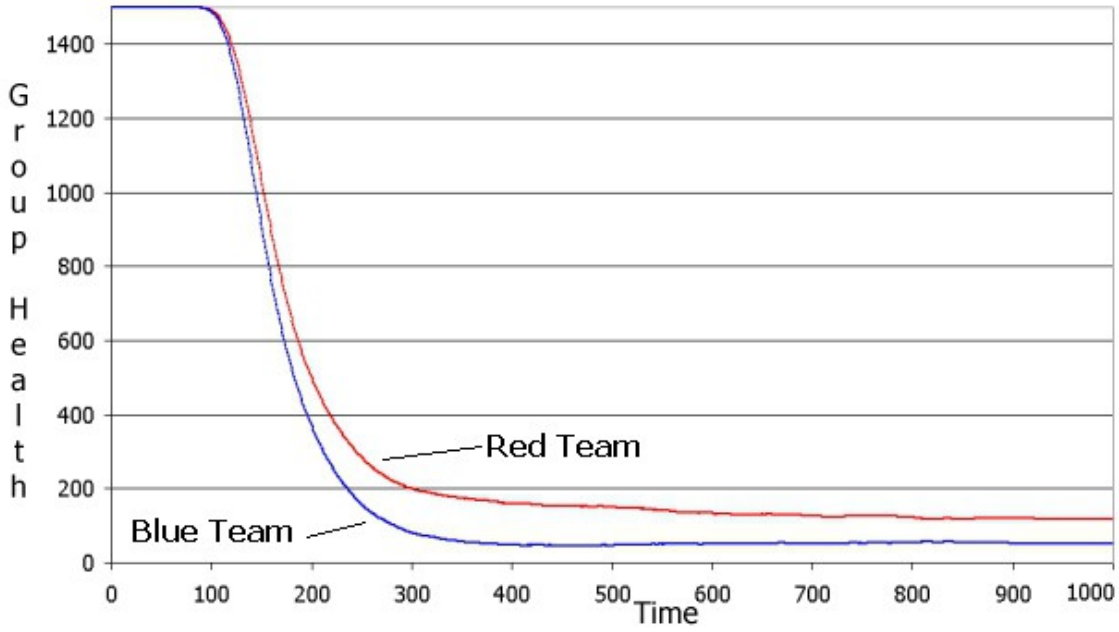


*Figure 5.8: Damage vs time plot for comms2 trial*

As can be seen in Figure 5.8, the Q-learning agents in *comms2* do significantly worse, than the *benchmark* when no communication takes place at all (Figure 5.6). This is also highlighted in Figure 5.10, as the Q-learning team does approximately 50 percent worse in the number of victories obtained using this communication method compared to the *benchmark*. This is despite the significantly increased learning rate shown for this run in Figure 5.12. The reason for this is the non-deterministic nature of a multi-agent system type of simulation. In the same state, the same action chosen may produce different rewards at different time steps. Thus by processing only the negative rewards, the actions producing them could be actions which might normally produce positive rewards at other times. Only negative reinforcement of these actions would tend to make them be selected less in future thereby restricting chances for positive reward. However, in the light of the significant gains made from the communication of exclusively positive rewards in *comms1*, the option needed investigating.

### 5.7.4 Comms3 Trial

Finally, the most promising results were gained when both negative and positive rewards were communicated in *comm3*. In this simulation trial, Q-learning agents communicated all learning events, whether associated with a positive or negative reward. The resulting damage vs. time plot is shown in Figure 5.9. The standard deviation value $\sigma$ for the average blue plot line in Figure 5.9 is 9.68.

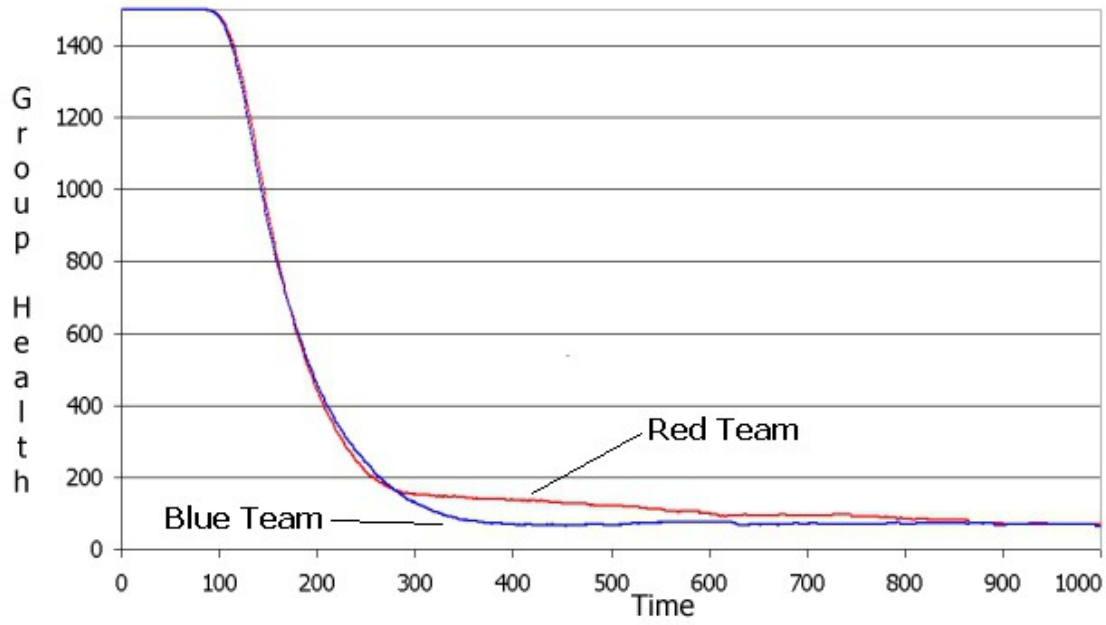


*Figure 5.9: Damage vs time plot for comms3 trial*

This represents a slightly better result than that shown in Figure 5.7, where learning events associated with only positive rewards were communicated. This shows an eight percent increase in the number of decisive victories, rising from 48 percent to 52 percent, and a corresponding drop in the victory percentage of the instinctual-agent team (red team) to just under 44 percent.

### 5.7.5 Initial Discussion

All of the above results have been summarized and collated into a comparative bar graph and shown in Figure 5.10. The red colored bars represent the number of wins in the red team (Instinctual agent team) over 1000 simulation runs for that trial. The blue

colored bars represent the number of wins of the blue team (Q-learning team) over 1000 simulation runs.

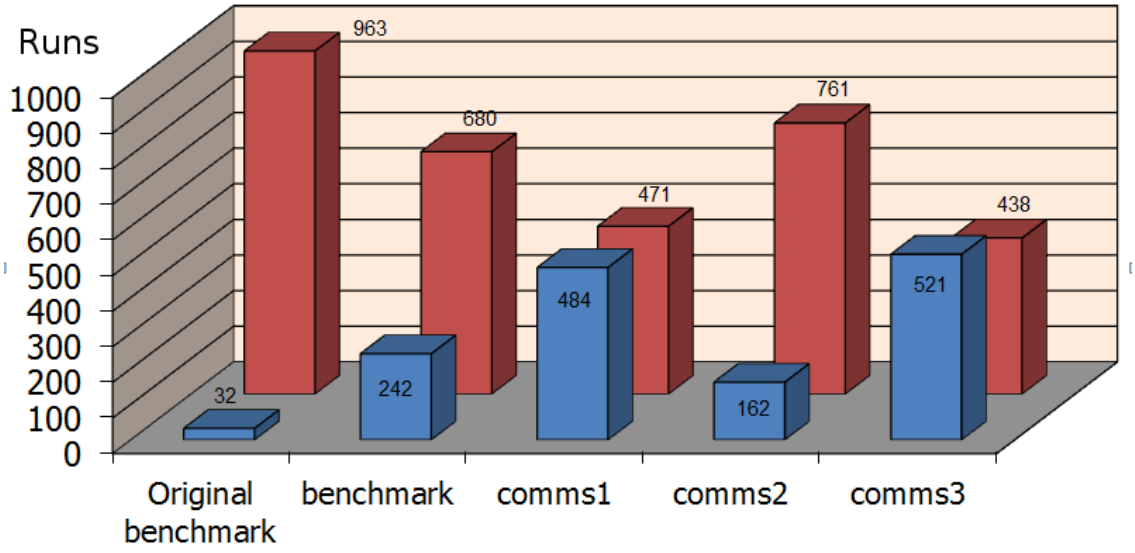


*Figure 5.10: Comparative bar graph summarizing Q-learning agent team wins vs Instinctual agent team wins for all trials*

In previous sections we have mentioned the significant difference between various trials, but significance between trials needs to be tested statistically before we can claim this as an achievement. Given the nature of the simulation it is difficult to use the data collected in some of the traditional tests for significance. Each trial consisted of over 1000 simulation runs, but each simulation run did not last for the same simulated time period. The length in time-steps for each run depended on the success or otherwise of the teams involved. The complex nature of simulation and the $\epsilon$ -greedy policy of randomly choosing an action based on the $\epsilon$ parameter in the reinforcement learning algorithm leads to a butterfly effect on the success of the blue team on a run-by-run basis.

By utilizing the simple trial data collected as shown in Figure 5.10 we can perform a significance test on two independent samples (trials) with a discrete outcome. This is similar in nature to testing reported success of a drug on two independent sample populations. There are a number different significance tests that can be utilized and

these include a 2 x 2 contingency table incorporating a Chi-Square statistic, or a Z-test for 2 population proportions. We can easily utilize the Z-test for 2 population proportions by comparing the count of blue team wins between different trials, as the red team remain fixed and the only variation is in the communication or functioning of the blue-team. The equation for calculating the Z score is given in Figure 5.11 (Stangroom 2014).

$$\frac{(\bar{p}_1 - \bar{p}_2) - 0}{\sqrt{\bar{p}(1 - \bar{p})\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}}$$

*Figure 5.11: Z Test for 2 population proportions (Stangroom 2014)*

I n Figure 5.11 $\bar{p}_1$ is the proportion from the first population, $\bar{p}_2$ is the proportion from the second population $n_1$ and $n_2$ are the sizes of the two populations and $\bar{p} = (x_1 + x_2)/(n_1 + n_2)$ where $x_1$ and $x_2$ are the number of successes in each population. The equation assumes the null hypothesis $H_0 : \bar{p}_1 = \bar{p}_2$ (there is no significant difference between the observed successes in the two populations) and the alternative hypothesis that $\bar{p}_1 \neq \bar{p}_2$ .

We use the results shown in Figure 5.10 for the number of blue team successes for the two populations, the *original benchmark* test and the second *benchmark* test involving the updated state space in the equation shown in Figure 5.11. Choosing a significance level $\alpha$ = 0.05 the Z score is -13.6565 and the p-value is 0.00001. Thus the result is significant for p < 0.05. We can then reject the null hypothesis on this basis.

Thus there is a significant difference between the two benchmark trials indicating the need to carefully design the state space and the set of possible actions in order for the reward function to properly drive the behavior of the agents. It is possible that

further refinement of the state space and action list may have produced further gains in the Q-learning team. However, the modified state space variables and actions list were not further modified for this research. The purpose of the research is to study the effects of communication on the Q-learning team performance.

The inclusion of the *original benchmark* results from Section 5.5 Initial Benchmark Results in Figure 5.10 above was as a comparison to highlight the difference between the later *benchmark* trial discussed in Section 5.7.1 Benchmark Trial and thus the importance of selection of the state space and available agent actions. In all future references to the *benchmark* trial, we will be referring to the *benchmark* trial conducted using the refined state space and actions list from Section 5.7.1 Benchmark Trial.

In examining the differences between the four trials discussed above (Table 5.7) which are highlighted in Figure 5.10, it must be remembered that the *red team* parameters were kept constant throughout all trials. So the differences in the results from the red team shown in Figure 5.10 are due solely to the different successes of the *blue team* (Q-learning team). The only difference between the experimental trials was in the learning events that were communicated. Based on the previous discussion on learning rates, we can examine the average learning rates of the group $l(X)$ for the Q-learning team for each of the simulation trials. These learning rates are shown in Figure 5.12. In all trials, the communication was global and no range restrictions on communication were enforced.

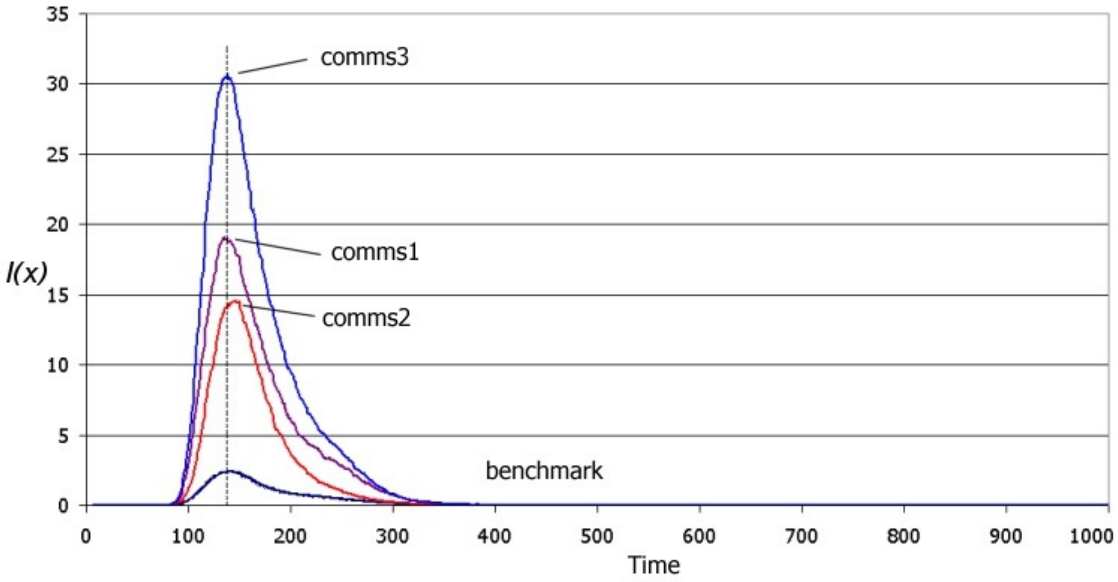


*Figure 5.12: Comparative group learning rates of the Q-learning team (blue team) for the four simulation trials*

The results shown in Figure 5.12 were somewhat anticipated, and demonstrate a significant difference in the learning rate between the Q-learning agent teams that communicate and the benchmark Q-learning team that does not. Interestingly enough, the learning rates for all four experimental runs peak at approximately the same time throughout the simulation. This was unexpected as it was assumed that an increased learning rate might bring forward the learning rate peak somehow. However, this may be due in part to the spatial positioning of the agents as the two groups meet and hostilities begin.

It is also worth noting that the peaks for the learning rate for both the *benchmark* and the *comms3* trials do not come close to their upper bound. From previous discussions in Section 5.2 Model for Co-operative Learning, the learning rate of the *benchmark* trial where the agents do not communicate is bounded by the linear function $l(X) = n$. With $|X|=15$ (n equal to 15), the peak learning rate is approximately 2 – 3 learning events per discrete time unit. Similarly, with the *comms3* trial where the agents communicate all learning events, the learning rate is bounded by the simple quadratic function $l(X) = n^2$. Again, $|X|=15$. thus $n^2 = 225$ yet the peak learning rate is approximately 31 learning events per discrete time units. The learning

rate will depend on many factors including the spacial arrangement of the agents in relation to each other, the net reward being other than zero, the current state (which dictates the possible actions available) and the proximity of the Q-learning agents to enemy agents which enables them to perform actions possibly leading to a reward.

We can directly map the learning rates observed in Figure 5.12 to the performance of the group overall which is observed in Figure 5.10. These improvements in agent performance echo observable effects in similar research described earlier when utilizing communication to share rewards.

In the *benchmark* trial, the Q-learning agents could only manage to survive as a group in approximately 24% of the simulation runs. By allowing the agents in the group to communicate their positive learning events in the *comms1* trial, the learning rate for the group was increased (Figure 5.12) and the group significantly improved its chances of survival (as in Figure 5.10) to approximately 48 percent, doubling its chances.

As a test of this significance we can again use the Z-test discussed previously. We use the results shown in Figure 5.10 for the number of blue team successes for the two populations, the *benchmark* trial and the *comms1* trial involving the updated state space in the equation shown in Figure 5.11. Again, choosing a significance level $\alpha = 0.05$ the Z score is -11.2532 and the p-value is 0.00001. Thus the result is again significant for $p < 0.05$. We can reject the null hypothesis indicating there is a significant difference. In testing for the difference between the *benchmark* and the *comms2* trial, choosing a significance level $\alpha = 0.05$ the Z score is 4.4555 and the p-value is 0.00001. Again we can reject the null hypothesis.

Thus, the *comms2* trial again indicates a significantly worse performance for group survival than the *benchmark*. The Q-learning team does approximately 50 percent worse in the number of victories compared to the *benchmark* trial, and this is despite the significantly increased learning rate for *comms2* shown in Figure 5.12. An explanation could lie in the non-deterministic nature of this type of simulation. In the same state, the same action chosen may produce different rewards at different time steps. Thus by

processing only the negative rewards, the actions producing them could be actions which might normally produce positive rewards at other times. By processing only the negative learning events, the negative reinforcement of these actions would tend to make them be selected less in future thereby restricting chances for positive reward. However, in the light of the significant gains made from the communication of positive learning events, the option needed investigating.

Figure 5.10 also shows a further increase in the Q-learning groups survival chances for the *comms3* trial. As indicated in Figure 5.12, *comms3* has the highest learning rate of all the simulation-runs, and includes both the positive and negative learning events. However, in light of the previous discussion it wasn't initially obvious that the highest learning rate would provide the best outcome.

Given these significant performance gains due to agent communication, it is now important for the underlying domain of the simulation that we determine which aspects of the simulation have been affected in order to realize the gains. That is, as time increases, what is affected by $M(t)$ to cause the observed increase in performance.

## 5.8 Investigating Simulation Artifacts

The purpose of this research is to investigate the effect $M(t)$ on $X$ and indeed the individual effects of $M_j(t)$ on each $x_j \in X=\{x_1, x_2, \ldots x_n\}$. However, before we attempt to investigate the effects of the communication on the Q-learning agents, we need to verify to a reasonable degree that there is indeed an effect on $X$, and the performance gains evident in Figure 5.10 are not realized elsewhere.

An initial concern was that the observable performance enhancement of the Q-learning agents might have been an artifact of the simulation itself brought about by $M_j(t)$. To this end, a number of simulation metrics were investigated in an attempt to pinpoint those areas of the simulation which may have been effected by communication and thus, accelerated learning. Given that the *comms3* simulation run resulted in the best

performance against the *benchmark*, only the simulation data of the Q-learning teams from these two trials have been used in the following comparative investigations for simulated artifacts.

### 5.8.1 Agent Actuation Rate

The first metric investigated was the actuation rate of the agents in the Q-learning team. That is, the accumulative measurement over time of the number of agents positively participating in the simulation. In the context of this simulation, an agent is positively participating if it is contributing to the reduction in health of the opposing team. The point of actuation of an agent is the time step within the simulation when it first inflicts damage on an opposing agent. If the actuation rate increased, then it is likely that more damage to the opposing team could be inflicted early, giving a positive advantage as the simulation continues. The actuation rate of *comms3* is charted against that of the *benchmark* and shown in Figure 5.13.

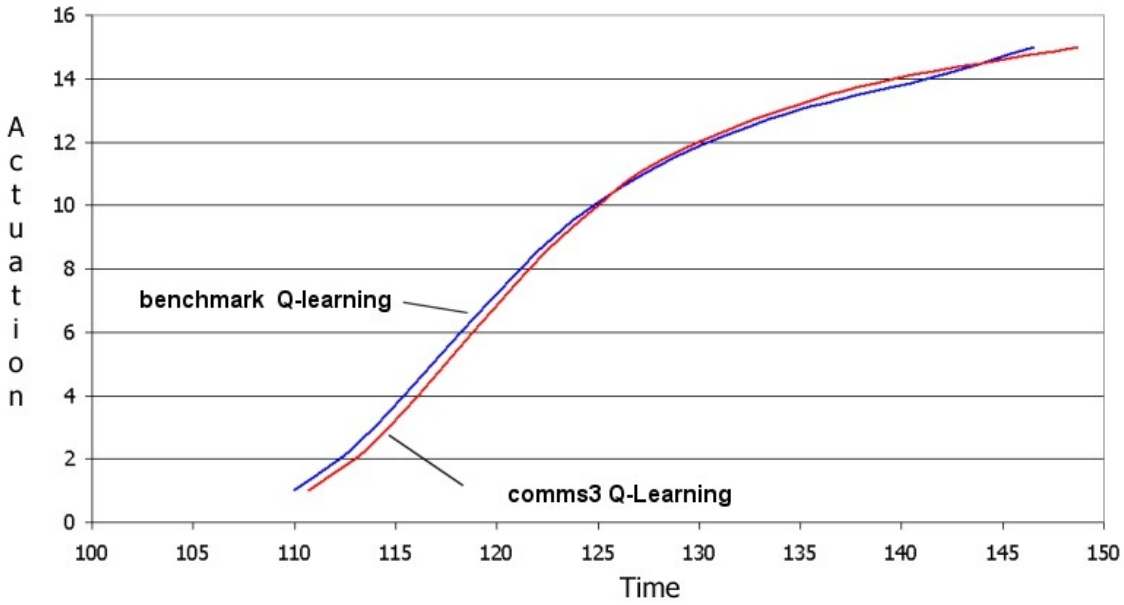


*Figure 5.13: Actuation rate of agents in the Q-learning team for the benchmark compared to the comms3 trial*

In Figure 5.13, the Horizontal axis represents the simulation time steps and the Vertical axis represents the accumulated number of agents positively active in the team.

In this figure the blue portion of the graph represents the actuation rate of the Q-learning agents in the *benchmark* trial. The red portion of the graph represents the actuation rate of the Q-learning agents in the *comms3* trial. The graphs represent the averaged results over all the simulations runs for each of the trials.

As apparent in Figure 5.13, there is little if any difference in the actuation rates between the two Q-learning teams. Thus, $M(t)$ and $l(X)$ (from Figure 5.12) do not seem to have affected this metric. As discussed earlier though, the spatial positioning of the agents within the environment may influence this to some degree, but it seems to remain unaffected. In this metric, all the agents from both teams begin to be positively active in the team learning within the short simulation time period of 110 to approximately 148.

Two other items investigated to see if performance increases could be due to simulation artifacts are the rate of the positive and negative rewards received from the simulation as time progresses. This could give an indication of the activity of the Q-learning group during this time period.

### 5.8.2 Reward Rates

It was postulated that perhaps the reward rate (receipt of positive or negative rewards) per simulated time step, which will have been influenced by the increased learning due to $M(t)$, could possibly alone be responsible for the increase in performance of $X$. To investigate this possibility, the rate of rewards received from the Q-learning team in the *benchmark* and *comms3* trials over the 1000 time steps of the experimental runs were graphed. This gives an indication of the activity of the Q-learning group during this time period. The reward rates of the positive and negative rewards were graphed separately.

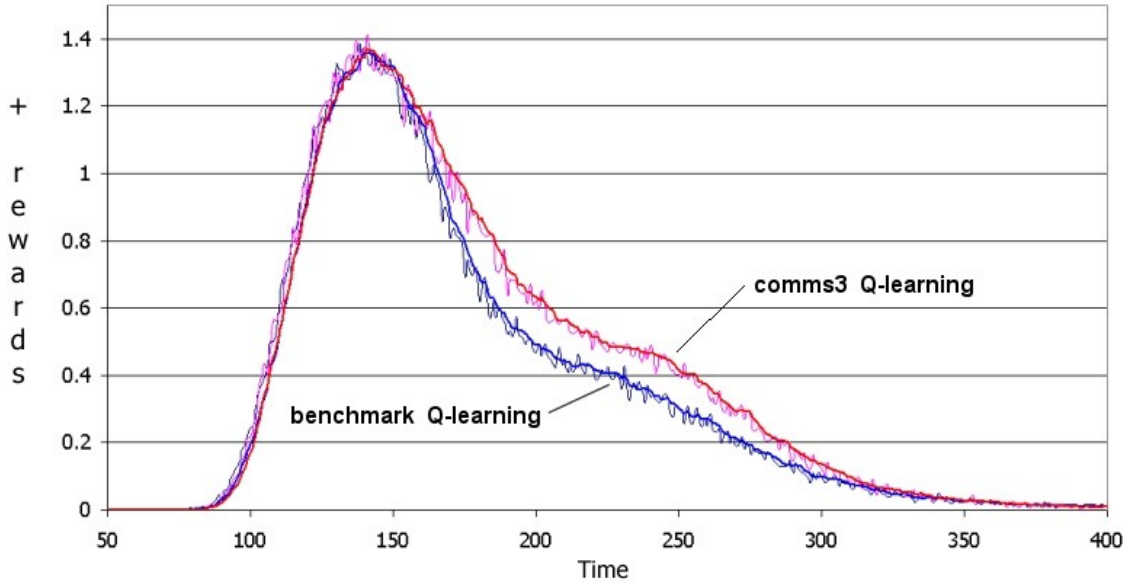


*Figure 5.14: Comparative plot of the rate of positive rewards received from the simulation for the Q-learning benchmark and comms3 trials*

In Figure 5.14, the Horizontal axis represents the simulation time steps and the Vertical axis represents the positive reward rate per simulation time step for individual agents. In this figure the blue portion of the graph represents the positive reward rate of the Q-learning agents in the *benchmark* trial. The red portion of the graph represents the positive reward rate of the Q-learning agents in the *comms3* trial. The graphs represent the averaged results over all the agents in all simulations runs for each of the trials.

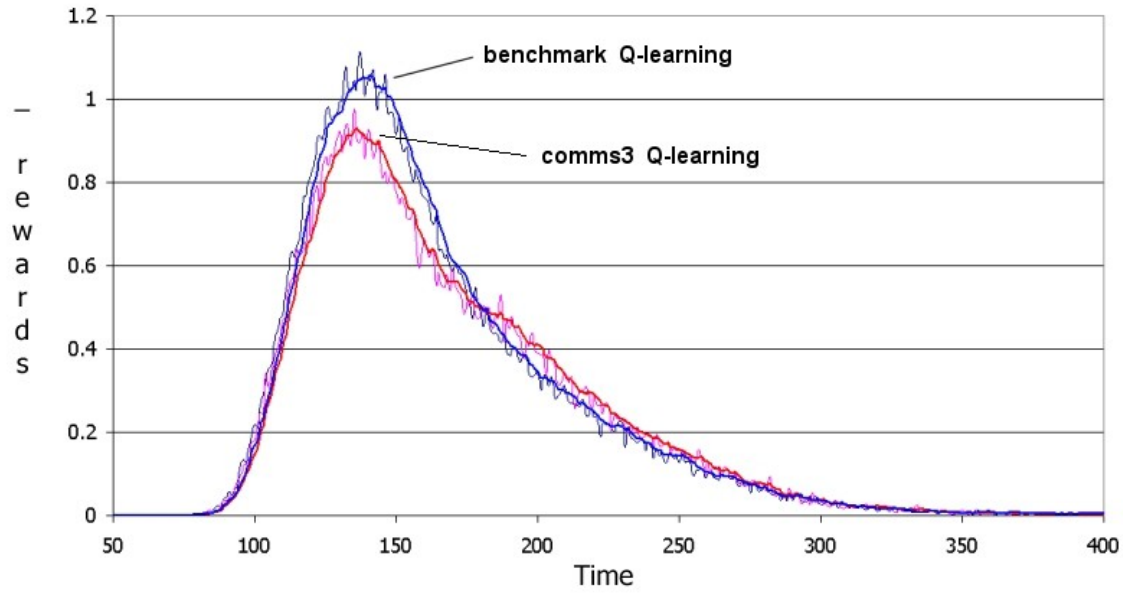


*Figure 5.15: Comparative plot of the rate of negative rewards received from the simulation for the Q-learning benchmark and comms3 trials*

In Figure 5.15, the Horizontal axis represents the simulation time steps and the Vertical axis represents the negative reward rate per simulation time step for individual agents. In this figure the blue portion of the graph represents the negative reward rate of the Q-learning agents in the *benchmark* trial. The red portion of the graph represents the negative reward rate of the Q-learning agents in the *comms3* trial. The graphs represent the averaged results over all the agents in all simulations runs for each of the trials.

Figure 5.14 and Figure 5.15 graph the reward rates (positive and negative respectively) during simulation time-steps 50-400. Time-steps outside this range were not graphed, as they didn't indicate any significant activity. There are observable differences in the reward rates. In Figure 5.14, after the apex in the positive reward rate, the *comms3* Q-learning team sustains a slightly higher rate of rewards for approximately 160 simulation time steps. In Figure 5.15, it can be observed that the *comms3* Q-learning team maintains a slightly lower rate of negative rewards for the first part of the conflict. However, there is a crossover at about time-step 180 where the *benchmark* Q-learning team maintains a slightly lower negative reward rate for some time.

The observable differences in the above figures do not seem significant. Thus it is questionable as to whether the accumulative effects of the increased positive reward and

the lower negative reward rates are enough to account for the differences in the final outcomes. Although the rise in the success rate of the Q-learning team is significant (Figure 5.10), we might be witnessing a possible '*Butterfly Effect*' due to the Complex Systems nature of the simulation. However, the answer may lay elsewhere. The metrics investigated above are simulation metrics, directly quantifiable by measuring agent behavior within the simulation. As the communication of learning events accelerates the agent's learning rates, we need to examine what the Q-learning agents themselves actually learn. By looking inside the agent we can see some significant and surprising differences/ similarities.

## 5.9 Examining Agent Knowledge Landscapes

In this section we will address the fundamental question of the research, exactly where the effects of communication are realized. We have conducted a number of experiments which are listed in Table 5.7 and discussed in *Section 5.7 Experimental trials*. These have shown that when agents communicate their learning experiences, then in general, they perform better, and hence the team of agents is more likely to succeed in their common goal, implicitly set by the reward function. This is consistent with previous research discussed in *Chapter 2 Literature Review*. In the previous section we investigated a number of simulation metrics to ensure that the performance gains are not a side-effect of the simulation brought about by the communication itself.

Given that there is a significant performance gain when agents communicate their learning events, then as discussed in *Chapter 1 Introduction*, the communication $M(t)$ at time *t* must have a discernible effect on the group $X=\{x_1, x_2, \ldots x_n\}$ at time *t+1*, $\Delta_{t+1} X$. Thus, more formally, the change in the group from time *t* to time *t+1* can be denoted as

$$\Delta_{t+1} X : \; X_t \overset{M(t)}{\rightarrow} X_{t+1}$$

The changes in $X$ from time $t$ to $t+1$, $\Delta_{t+1} X$, will be realized in the individual agents whose total effect will be $\Delta_{t+1} X$. That is

$$\forall x_i \in X, \quad \Delta_{t+1} X = \{\delta_{t+1} x_i\}$$

where $\delta_{t+1} x_i$ are the resultant changes in $x_i$ at time $t+1$ caused by $M_i(t)$. How then do we observe the individual changes $\delta_{t+1} x_i$? Observing these changes is difficult, and can only be done on an individual basis by examining minutely the state-action matrix of the *Q[]* array, as shown in Figure 5.16. The *Values[]* vector will contain the accumulated values of the new reinforcement algorithm through the application of Equation (5.2). However, Equation (5.2) works by first updating an agents state-action matrix from its own learning event (if any) and then continues to update the state-action matrix utilizing the learning events communicated from other agents. The only way to discern the effects of communication by studying the individual state-action matrices would be to '*pull apart*' Equation (5.2). That is, update the agents state-action matrix from its own learning event, note the results, then apply the remainder of Equation (5.2) and update the state-action matrix from the shared learning events from the agents team members, then note the difference.

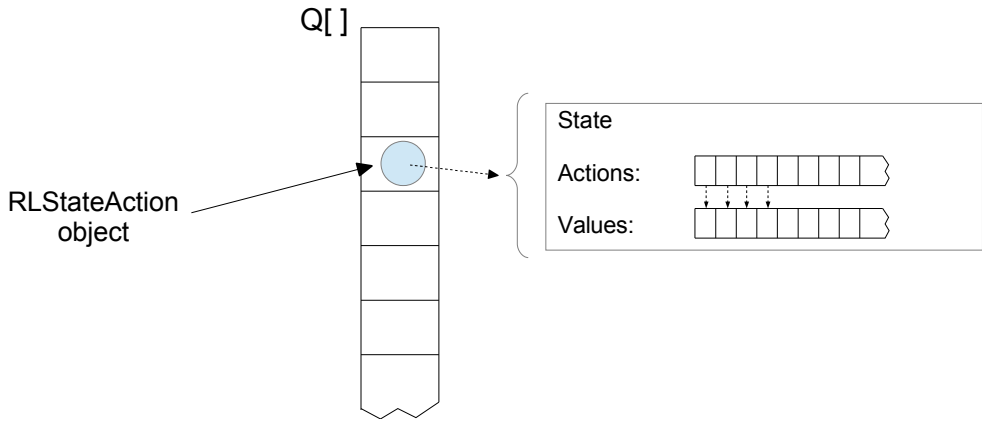


*Figure 5.16: Implementation of Q, state-action matrix (from Chapter 4 Section 4.5 The Thinking Cycle)*

This would still not yield a complete picture due to the non-markov nature of the multi-agent simulation. The learning algorithm does not converge and the optimal value

function becomes a moving target. The changes to the individual agents $\{\delta_t x_i\}_{t=1\ldots T}$ are not consistent between simulation runs and trying to discern the effects of communication in this manner is not a satisfactory approach. A more complete picture can be gained by examining groups of agents from each of the four different simulation trials at each end of the success spectrum. That is, in each simulation trial, examine the most successful agents, and the least successful agents and compare these across the different simulation trials. In this way we can develop a better understanding of the overall effects of communication on the group and on the individual agents.

In this research we have developed what we term the knowledge landscape to help provide a visual window into what the agents have learned. By examining the knowledge landscape and analyzing the state-action matrices, we can build an understanding of $\{\Delta_t X_i\}_{t=1\ldots T}$ . The Knowledge landscape is constructed by plotting the values in the Q-learning agents state-action matrix as a three-dimensional surface graph. The state-action matrix *Q*[] for each agent consists of an array of state-action objects which contain vectors of different lengths to hold values for the valid actions of a particular state as shown in Figure 5.16. Thus the lengths of the *Actions*[] and *Values*[] vectors differ from state to state due to the possible differences in number of actions per state, as shown in the *State Action Map* listed in *Appendix B.1*. The knowledge landscape graphs are developed with a set 68 x 11 base matrix (68 possible states and 11 possible actions). The values from individual agents state-action matrices are mapped onto this fixed array for visual display purposes.

To our knowledge, such graphing techniques have not been used before to view the accumulated knowledge of an agent as a whole surface plot. Viewing the state-action matrix in this fashion allows us to observe with a coarse-grain view the overall shape of the accumulated knowledge, showing the peaks where positive rewards have accumulated and we can pinpoint specific actions or groups of actions that the agent finds successful. Successful in this context means that the actions will most likely lead to a positive reward. Similarly, we can additionally observe the troughs which pinpoints specific actions or groups of actions that the agents do not find successful, that is, will most likely lead to a negative reward.

When viewing the three-dimensional knowledge landscape diagrams, the horizontal axis represents the different possible states an agent can be in. The depth axis on the three-dimensional graph represents the 11 possible actions that can be chosen, though not all actions are applicable to all states. The value in any cell of the base matrix of a knowledge landscape diagram represents the accumulated reward function values gained through the application of Equation (5.1) or Equation (5.2) as the simulation progresses through time, depending on whether we are examining the *benchmark* trial, or the *comms1*, *comms2* or *comms3* trial results. The vertical axis being the third dimension of the knowledge landscape diagram, represents the graphing along the *z-axis* of the values contained in the surface plots base matrix..

The knowledge landscape diagrams allow us to view where the major areas of learning activity have accumulated as either positive or negative effects on the agent's state-action matrix. Examining the knowledge landscape of a single agent may be useful when studying a single simulation, however, in a complex system this would not be indicative of the overall learning for a comparative trial. To visualize the discernible difference, $\{\Delta_t X_i\}_{t=1\ldots T}$, in the agents for a particular trial, all 2000 simulation runs of the trial are scanned, and the five most successful agents and five least successful agents in each simulation run are recorded. The five least successful agents in a simulation run are the first to die, and the five most successful agents the are last to die, or the five left alive in the simulation with the highest health points left. The state-action matrices of the five most successful agents over all 2000 simulation runs in the trial are then averaged and recorded, as are the state-action matrices of the five least successful agents over all 2000 simulation runs in the trial. These are then used to produce two knowledge landscape diagrams for each simulation trial, one for the most successful agents in the trial, and one for the least successful agents in the trial. Comparing these two knowledge landscape diagrams allows us to compare the average differences between the successful and unsuccessful agents in a trial, providing visual data on $\{\Delta_t X_i\}_{t=1\ldots T}$. We can then compare the different knowledge landscape diagrams between the different trials which provides further evidence.

Along with the visual data provided by the knowledge landscape diagrams, we can then analyze individual state-action matrices between agents in each end of the success spectrum to see the results.

*Note: In the following pages the Knowledge Landscape diagrams are presented in a rotated landscape format so a larger version of the diagram can be shown. Thus, detail is not lost in minimization. As a consequence some pages will have blank areas to keep the discussion in synch with the diagrams.*

### 5.9.1 Benchmark Trial

The knowledge landscape diagram for the least successful agents in the benchmark trial is shown below in Figure 5.17. As discussed in the previous section, the horizontal axis represents all 68 possible states, and the depth axis represents all 11 possible actions. The vertical (or z) axis represents the value from the corresponding state-action matrix for the specific state-action cell.

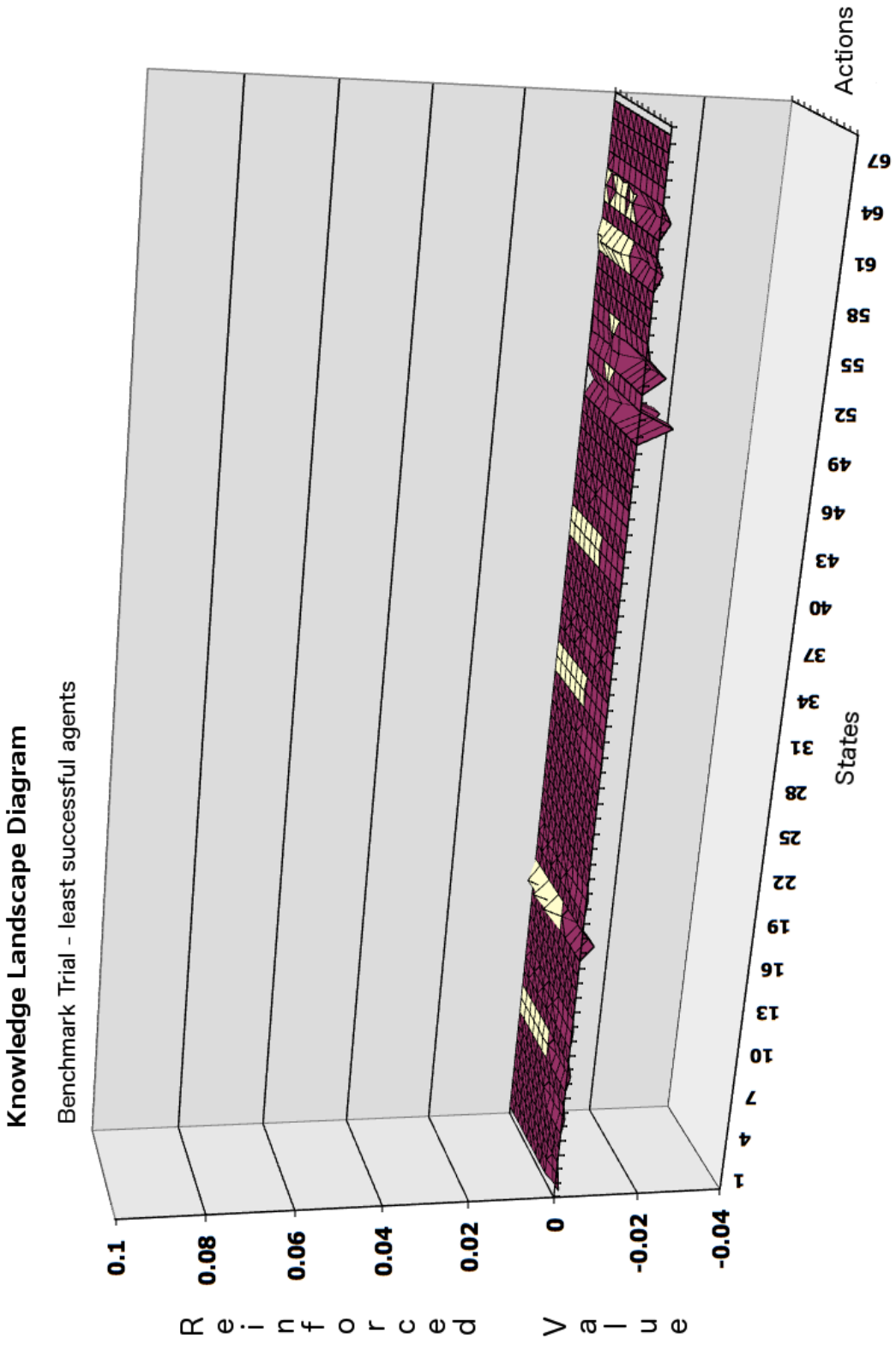


*Figure 5.17: Benchmark trial knowledge landscape diagram - least successful agents*

With the benchmark trial, no communication between the agents takes place, thus they are all individual learners. The knowledge gained throughout the simulation is gained solely from the agents own learning events. As can be seen in Figure 5.17, there is little in the way of consolidation of actions that lead to rewards. This can be contrasted with the knowledge landscape diagram for the most successful agents in the benchmark trial shown in Figure 5.18. While there is some minor learned responses in states 9-26, representing states when there is enemy in both visual and firing range, the most reinforced actions lie within states 51-64. This represents the states where both enemy and wounded enemy are in visual and firing range. These states would be more common towards the latter half of the combat engagement, thus it is not unexpected that these states would not be reinforced much in the early stages of the simulation, corresponding to times the least successful agents would have been active. The most successful agents would remain in the simulation long enough to have tried these actions and have received a positive reward.

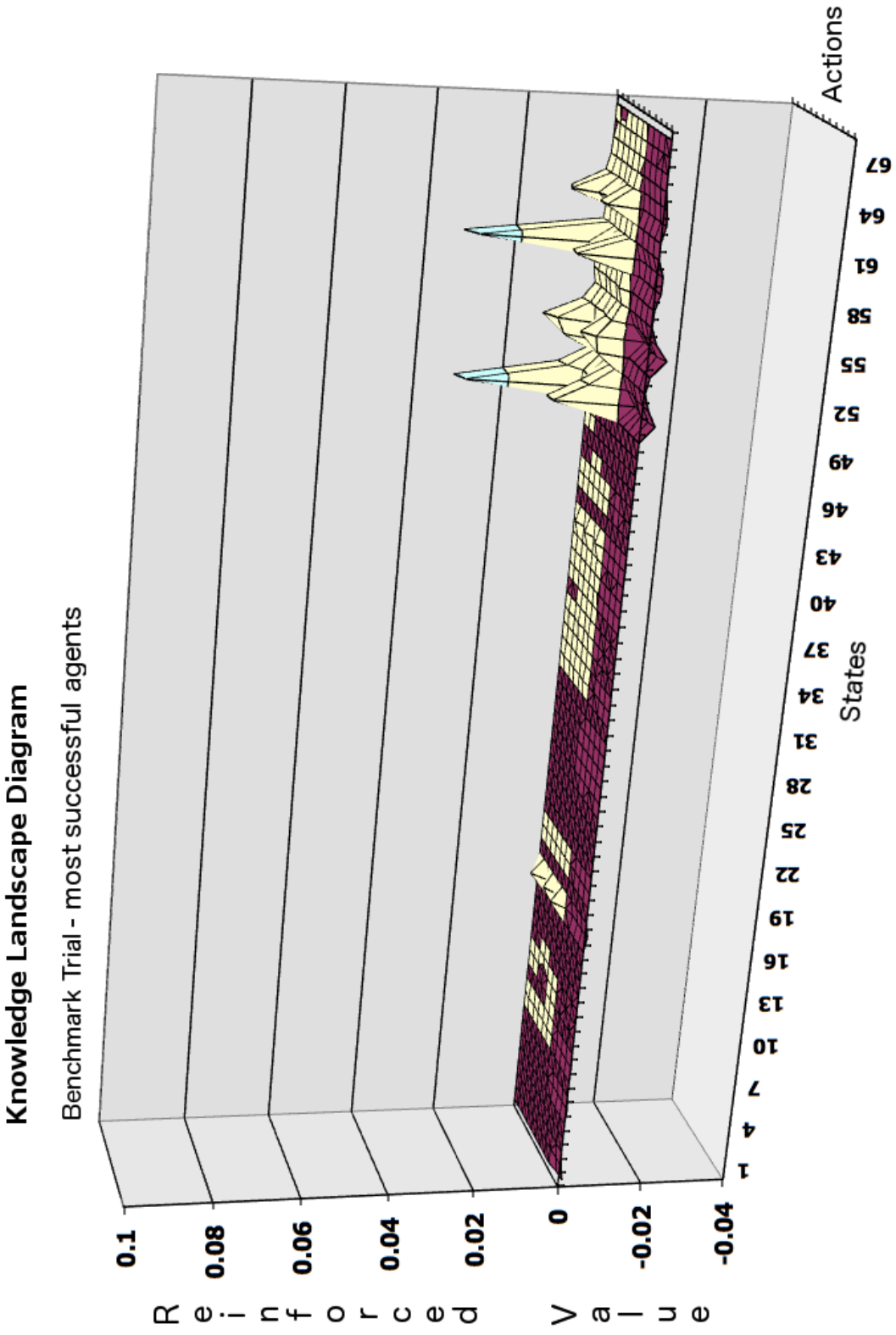


*Figure 5.18: Benchmark trial knowledge landscape diagram - most successful agents*

The actions associated with the states containing more consolidated knowledge in Figure 5.18 are actions six to eleven (6 – 11) which are the actions involving moving towards wounded enemy, firing on wounded enemy and actions involving moving and firing. The two tall consolidated peaks in states 51, 52 and again in states 60 and 61 are for the actions moving away and firing at the enemy (movawayF), and moving away and firing at wounded enemy (movawayFW). Interestingly, in these states the state variable *visualEFRatio*, is set to 0, which means there are more friendly agents in visual range.

We can compare the state-action matrices of two individual agents from the *benchmark* trial. In simulation run 271 in the benchmark trial, agent 27 is the least successful agent, dying early in the simulation. The listing of the agents state-action matrix is shown in Listing 5.2 and Listing 5.3 below. The listing was broken into two separate listings for formatting reasons.

```
state 0 0.0  0.0
state 1 0.0  0.0
state 2 0.0  0.0  0.0  0.0
state 3 0.0  0.0  0.0  0.0
state 4 0.0  0.0  0.0  0.0
state 5 0.0  0.0  0.0  0.0
state 6 0.0  0.0  0.0  0.0
state 7 0.0  0.0  0.0  0.0
state 8 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 9 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 10 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 11 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 12 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 13 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 14 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 15 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 16 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 17 0.0  -0.095  0.0  0.0  -0.0050000000000000044  0.0  0.0
state 18 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 19 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 20 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 21 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 22 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 23 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 24 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 25 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 26 0.0  0.0  0.0  0.0  0.0
state 27 0.0  0.0  0.0  0.0  0.0
state 28 0.0  0.0  0.0  0.0  0.0
state 29 0.0  0.0  0.0  0.0  0.0
state 30 0.0  0.0  0.0  0.0  0.0
state 31 0.0  0.0  0.0  0.0  0.0
state 32 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 33 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.2: State-action matrix of agent agent 27 in simulation run 271 – states 0 to 33

The following listing completes the state-action action matrix for agent 27 showing states 34 to 67.

```
state 34 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 35 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 36 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 37 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 38 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 39 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 40 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 41 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 42 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 43 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 44 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 45 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 46 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 47 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 48 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 49 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 50 0.0  -0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 51 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 52 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 53 -0.05  0.0  0.0  0.0  0.0  0.0  0.0  0.0  -0.004500000000000004  0.0  0.0
state 54 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 55 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 56 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 57 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 58 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 59 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 60 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 61 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 62 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 63 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 64 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 65 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 66 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 67 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.3: State-action matrix of agent agent 27 in simulation run 271 – states 34 to 67

As can be seen in Listing 5.2 and Listing 5.3 above, very little in the way of accumulated values due to the application of Equation (5.1) can be seen. The values indicate only negative reinforcement, and mainly for actions 2 and 3 in states 50 and 53. This agent was killed before there were enough positive rewards to positively reinforce enough state-actions to provide enough actions which might lead to a positive reward. It must be remembered that agent 27 is an individual agent and the knowledge landscape diagram in Figure 5.17 represents an average of the least successful agents over 2000 simulation runs.

In contrast to the state-action matrix for agent 27, we can examine the state-action matrix of one of the most successful agents in the benchmark trial. In simulation run 271 in the *benchmark* trial, agent 19 is the most successful agent, remaining alive in the

simulation with the most health points. The listing of the agents state-action matrix is shown in Listing 5.4 and Listing 5.5 below. Again, the listing was broken into two separate listings for formatting reasons.

```
state 0 0.0  0.0
state 1 0.0  0.0
state 2 0.0  0.0  0.0  0.0
state 3 0.0  0.0  0.0  0.0
state 4 0.0  0.0  0.0  0.0
state 5 0.0  0.0  0.0  0.0
state 6 0.0  0.0  0.0  0.0
state 7 0.0  0.0  0.0  0.0
state 8 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 9 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 10 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 11 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 12 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 13 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 14 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 15 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 16 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 17 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 18 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 19 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 20 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 21 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 22 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 23 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 24 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 25 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 26 0.0  0.0  0.0  0.0  0.0
state 27 0.0  0.0  0.0  0.0  0.0
state 28 0.0  0.0  0.0  0.0  0.0
state 29 0.0  0.0  0.0  0.0  0.0
state 30 0.0  0.0  0.0  0.0  0.0
state 31 0.0  0.0  0.0  0.0  0.0
state 32 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 33 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.4: State-action matrix of agent agent 19 in simulation run 271 – states 0 to 33

As can be seen in Listing 5.4, there is no accumulated positive reinforcement of any actions in states 0 - 33 for agent 19 in the state-action matrix. In states 34 – 67 shown in Listing 5.5 , there is some positive reinforcement for some actions in states 50, 53, 54, 56 and 57. These correspond to states in which there are both enemy and wounded enemy in visual and firing range, and the actions generally correspond to actions involving moving and firing at wounded enemy. These states also can be mapped to the area of consolidation which can be seen in the knowledge landscape diagram shown in Figure 5.18 for states 50 through 64. Again, the knowledge landscape diagram in Figure 5.18 represents an average of the most successful agents in the *benchmark* trial over 2000 simulation runs, while the state-action matrix in Listing 5.4 and Listing 5.5 is for an individual agent.

```
state 34 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 35 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 36 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 37 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 38 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 39 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 40 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 41 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 42 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 43 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 44 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 45 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 46 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 47 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 48 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 49 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 50 0.0  0.0  0.0  0.0  0.0  -0.05  0.0  0.0  0.05  0.0  0.0
state 51 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 52 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 53 0.0  0.0  0.0  0.0  0.05  0.0  0.0  0.0  0.0  0.0  0.0
state 54 0.0  0.0  0.0  0.0  -0.0045000000000000004  0.0  0.0  -0.05  -0.05  0.0  0.17458355
state 55 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 56 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0505
state 57 0.0  0.0  0.0  0.0  0.20726033050000003  0.0  0.0  0.0  0.0  0.0  0.0
state 58 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 59 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 60 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 61 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 62 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  -0.05  0.0
state 63 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 64 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 65 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 66 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 67 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.5: State-action matrix of agent agent 19 in simulation run 271 – states 34 to 67

Comparing these two knowledge landscape diagrams gives us some idea of what knowledge the agents might need to acquire in order to survive within the simulation. In the benchmark trial, the reinforcement learning agents only register a success in approximately 24 percent of simulation runs, and the margin is slim in terms of remaining health points for the surviving agents left alive at the end of the simulation.

### 5.9.2 Comms3 Trial

In this section we present the knowledge landscape diagrams for the *comms3* trial. In this trial, the agents communicate their learning experience with other agents in the group when both positive and negative rewards have been received. These are based on actions undertaken in the previous discrete time frame, see *Section 5.7 Experimental trials*. By comparing the knowledge landscapes between the *benchmark* and *comms3* trials we will observe the effects of communication, $\{\Delta_t X_i\}_{t=1...T}$ more dramatically. Though the knowledge landscapes of the *comms1* and *comms2* trial will be examined in the following sections for completeness.

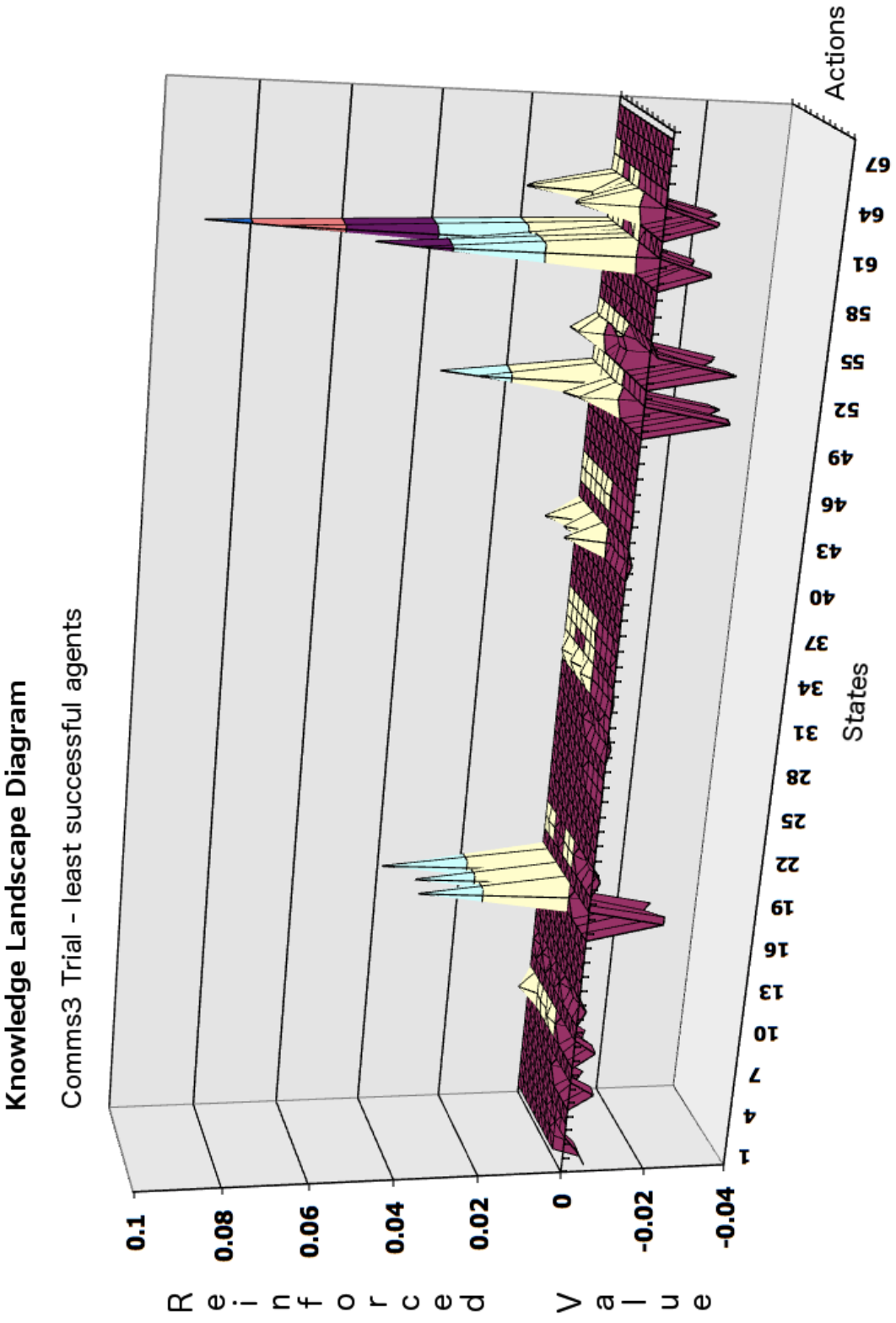


*Figure 5.19: Comms3 trial knowledge landscape diagram - least successful agents*

Figure 5.19 shows the knowledge landscape diagram for the least successful

agents in the *comms3* trial. Again, the knowledge landscape diagram represents the averaging of the state-action matrices of the five least successful agents in each of the 2000 simulation runs of *comms3*. A more detailed comparison between the *benchmark* and *comms3* trials will be made in *Section 5.10 Discussion*, but it can be seen that there is a marked difference between the knowledge landscape diagrams of the least successful agents in the *benchmark* and the least successful agents in *comms3*. It can be observed in Figure 5.19 that there is some significant positive reinforcement to actions 6, 8 10 and 11 in states 17, 18, 52, 53, 60, 61, 63 and 64. These correspond to states where enemy is in both firing range and visual range, and the actions involve firing on the enemy and movement to the enemy. There is some minor positive reinforcement in some other states, but the ones mentioned are significant in comparison. There is also some strong negative reinforcement in actions 2, 3, 4 and 5 in states 17, 8, 51, 52, 54, 55,60, 61, 63 and 64. The actions in question are the first five actions listed in the *State Action Map* in *Appendix B.1:* do nothing; explore; move away from the enemy locus; move to the closest enemy; fire.

The knowledge landscape diagram for the most successful agents in the comms3 trial is shown in Figure 5.20. For the most successful agents compared to the least successful agents, there is further consolidation with positive rewards for actions 6 -11 in states 51 and 53. The positive peaks here are much more pronounced and indicate that these actions have been successful and led to more positive rewards, further reinforcing these actions for future selection. There has also been consolidation of these actions in states 54 and 55 and again further reinforcement of the successful actions in states 60, 61 and states 63, 64. There has also been some positive reinforcement evident for actions 6-11 in states 30, 46, and while this is only slight it is still evident from visual observation when comparing the most successful agents to the least successful in *Comms3* in Figure 5.20 and Figure 5.19. Additionally, there is further negative reinforcement of the existing negative peaks in states 50 – 52.

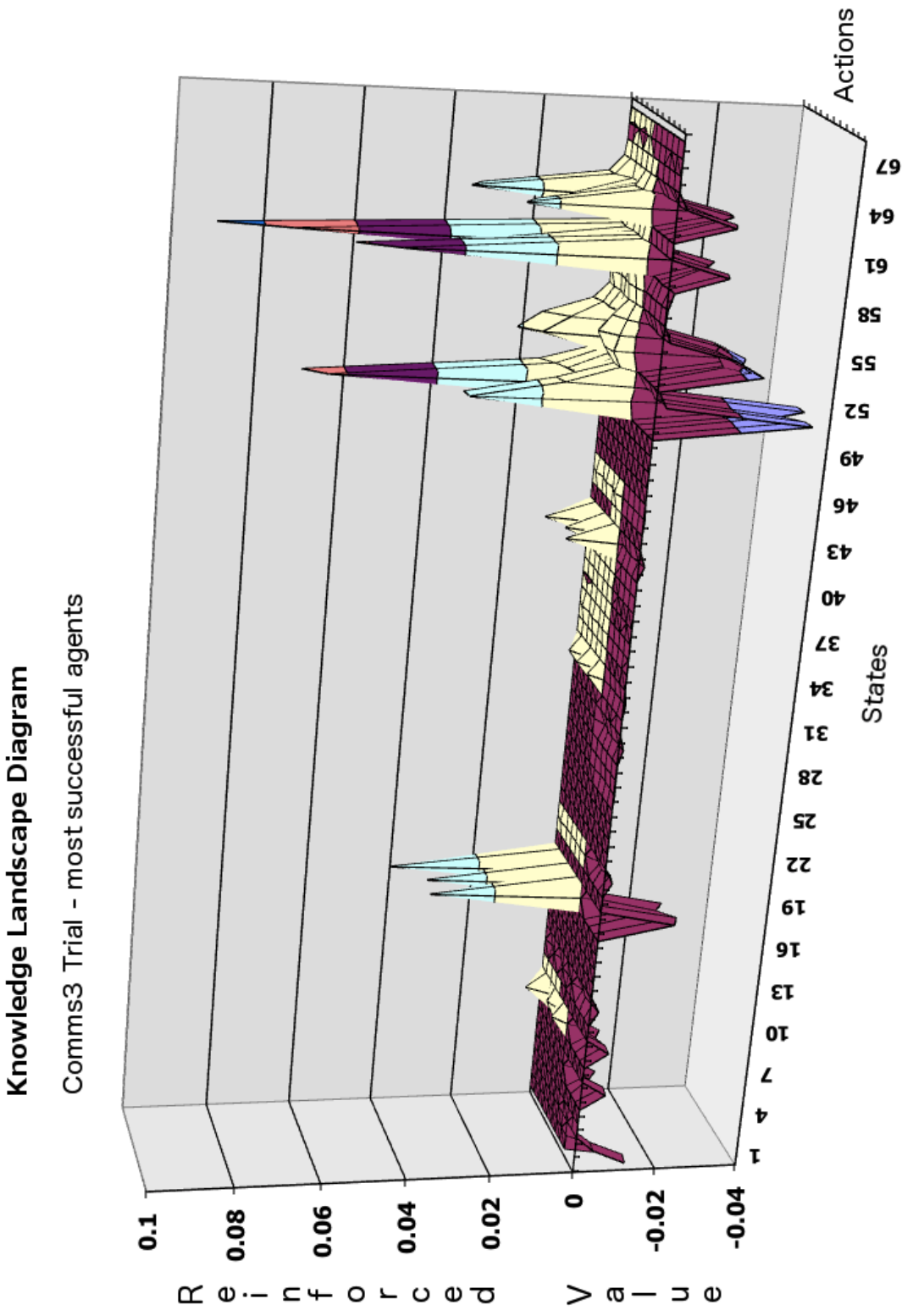


*Figure 5.20: Comms3 trial knowledge landscape diagram - most successful agents*

One of the most interesting results which will be discussed in *Section 5.10 Discussion,* can be seen when visually comparing the two knowledge landscape diagrams of the C*omms3* trial. When comparing the knowledge landscape diagrams in Figure 5.19 and Figure 5.20 it can be observed that there is very little difference between the two. Apart from further positive and negative reinforcement of existing peaks in Figure 5.19 (least successful agents), the two are surprisingly close. This is in contrast to the observable differences in the knowledge landscape diagrams between the least successful and most successful agents in the benchmark trial (Figure 5.17 and Figure 5.18).

```
state 0 0.0  0.0
state 1 0.0  0.0
state 2 0.0  0.0  0.0  0.0
state 3 0.0  0.0  0.0  0.0
state 4 0.0  0.0  0.0  0.0
state 5 0.0  0.0  0.0  0.0
state 6 0.0  0.0  0.0  0.0
state 7 0.0  0.0  0.0  0.0
state 8 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 9 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 10 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 11 0.0  0.0  0.0  0.0  0.0  -0.05  0.0
state 12 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 13 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 14 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 15 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 16 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 17 0.0  -0.05  -0.095  0.0  0.05  0.05  0.095
state 18 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 19 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 20 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 21 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 22 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 23 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 24 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 25 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 26 0.0  0.0  0.0  0.0  0.0
state 27 0.0  0.0  0.0  0.0  0.0
state 28 0.0  0.0  0.0  0.0  0.0
state 29 0.0  0.0  0.0  0.0  0.0
state 30 0.0  0.0  0.0  0.0  0.0
state 31 0.0  0.0  0.0  0.0  0.0
state 32 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 33 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.6: State-action matrix of agent agent 26 in simulation run 203 – states 0 to 33

As with the *benchmark* trial, we can compare the state-action matrices of two individual agents from the *comms3* trial and compare these with the knowledge landscape diagrams. In simulation run 203 in the *comms3* trial, agent 26 is one of the least successful agents, again, dying relatively early in the simulation. The listing of the agents state-action matrix is shown in Listing 5.6 and Listing 5.7. The listing was

broken into two separate listings for formatting reasons.

The following listing completes the state-action action matrix for agent 26 showing states 34 to 67

```
state 34 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 35 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 36 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 37 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 38 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 39 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 40 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 41 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 42 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 43 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 44 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 45 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 46 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 47 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 48 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 49 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 50 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0410370145258  0.0513693232469  0.1369323246993  0.0
state 51 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 52 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 53 0.0  0.0  0.0  0.0  0.0  -0.05  0.044992920330201025  0.0  -0.05  -0.021152560721388443  0.0
state 54 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 55 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 56 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 57 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 58 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 59 0.0  0.0  -0.05  0.0  0.0  0.0  0.0  0.05  0.051369050000000006  0.2088710305  0.0
state 60 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 61 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 62 0.0  -0.095  -0.095  -0.05  -0.01267695000000  -0.135  0.0  0.0  0.0607118297660  -0.05  -0.095
state 63 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 64 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 65 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 66 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 67 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.7: State-action matrix of agent agent 26 in simulation run 203 – states 34 to 67

As can be seen in Listing 5.6 and Listing 5.7, the state-action matrix for this agent shows much more activity in terms of positive and negative reinforcement than a corresponding least successful agent in the benchmark trial (see Listing 5.2 and Listing 5.3). The areas of activity in terms of actions and states experiencing reinforcement correspond to the areas of activity shown in the knowledge landscape diagram in Figure 5.19.

In comparison to an unsuccessful agent in the *comms3* trial, we can now examine the state-action matrix of one of the most successful agents in the *comms3* trial. In simulation run 203 in the *comms3* trial, agent 15 is the most successful agent, remaining

alive throughout the simulation with more health points remaining than the other agents. The listing of the agents state-action matrix is shown in Listing 5.8 and Listing 5.9 below. Again, the listing was broken into two separate listings for formatting reasons.

```
state 0 0.0  0.0
state 1 0.0  0.0
state 2 0.0  0.0  0.0  0.0
state 3 0.0  0.0  0.0  0.0
state 4 0.0  0.0  0.0  0.0
state 5 0.0  0.0  0.0  0.0
state 6 0.0  0.0  0.0  0.0
state 7 0.0  0.0  0.0  0.0
state 8 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 9 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 10 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 11 0.0  0.0  0.0  0.0  0.0  -0.05  0.0
state 12 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 13 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 14 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 15 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 16 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 17 0.0  0.0  -0.095  0.0  0.05  0.05  0.095
state 18 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 19 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 20 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 21 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 22 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 23 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 24 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 25 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 26 0.0  0.0  0.0  0.0  0.0
state 27 0.0  0.0  0.0  0.0  0.0
state 28 0.0  0.0  0.0  0.0  0.0
state 29 0.0  0.0  0.0  0.0  0.0
state 30 0.0  0.0  0.0  0.0  0.0
state 31 0.0  0.0  0.0  0.0  0.0
state 32 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 33 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.8: State-action matrix of agent agent 15 in simulation run 203 – states 0 to 33

As can be seen in Listing 5.8, there is some reinforcement of actions for states 11 and 17. The listing also confirms the observation from comparison of the two knowledge landscape diagrams for *comms3* that there is little different between the least and most successful agents. This is further confirmed by an examination of the remaining states (34 – 67) in the agents state-action matrix shown in Listing 5.9. This shows consolidation of reinforcement for similar actions as that of the least successful agent shown in Listing 5.6 and Listing 5.7. These areas of reinforcement can also be mapped to the areas of reinforcement in the knowledge landscape diagram presented in Figure 5.20.

```
state 34 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 35 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 36 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 37 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 38 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 39 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 40 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 41 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 42 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 43 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 44 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 45 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 46 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 47 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 48 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 49 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 50 0.0  0.0  0.0  0.0  0.09873517895842833  0.0  0.24202315667337493  -0.01484681850908931
0.10067299726043928  0.07840306179321378  0.052228000526637314
state 51 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 52 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 53 0.0  0.0  0.0  0.0  0.0  -0.05  0.05659831757636667  0.0  -0.05  -0.037539682242334185  0.0
state 54 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 55 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 56 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 57 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 58 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 59 0.0  0.0  -0.05  0.0  0.0  0.0  0.0  0.05  0.051369050000000006  0.24211671690000586  0.0
state 60 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 61 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 62 0.0  -0.095  -0.095  -0.05  -0.013045500000002  -0.135  0.0  0.0  0.060839501018282 -0.05  -0.095
state 63 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 64 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 65 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 66 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 67 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

Listing 5.9: State-action matrix of agent agent 15 in simulation run 203 – states 34 to 76

The strong similarity of the two state-action matrices shown above will be discussed in Section 5.10 Discussion, but is a consequence of the communication between agents in the simulation.

### 5.9.3 Comms1 Trial

In the *comms1* trial, agents were allowed to communicate their learning events, but only those that led to a positive reward. It was initially postulated that by only communicating positive learning experiences, the agents will very quickly accumulate a reinforcement of many actions that lead to positive rewards, giving them a better opportunity to choose these actions and thus to survive. While the agents do better than those which do not communicate their learning events, they do not do as well as those which communicate all learning events. The knowledge landscape diagram for the least successful in the *comms1* trial is shown in Figure 5.21.

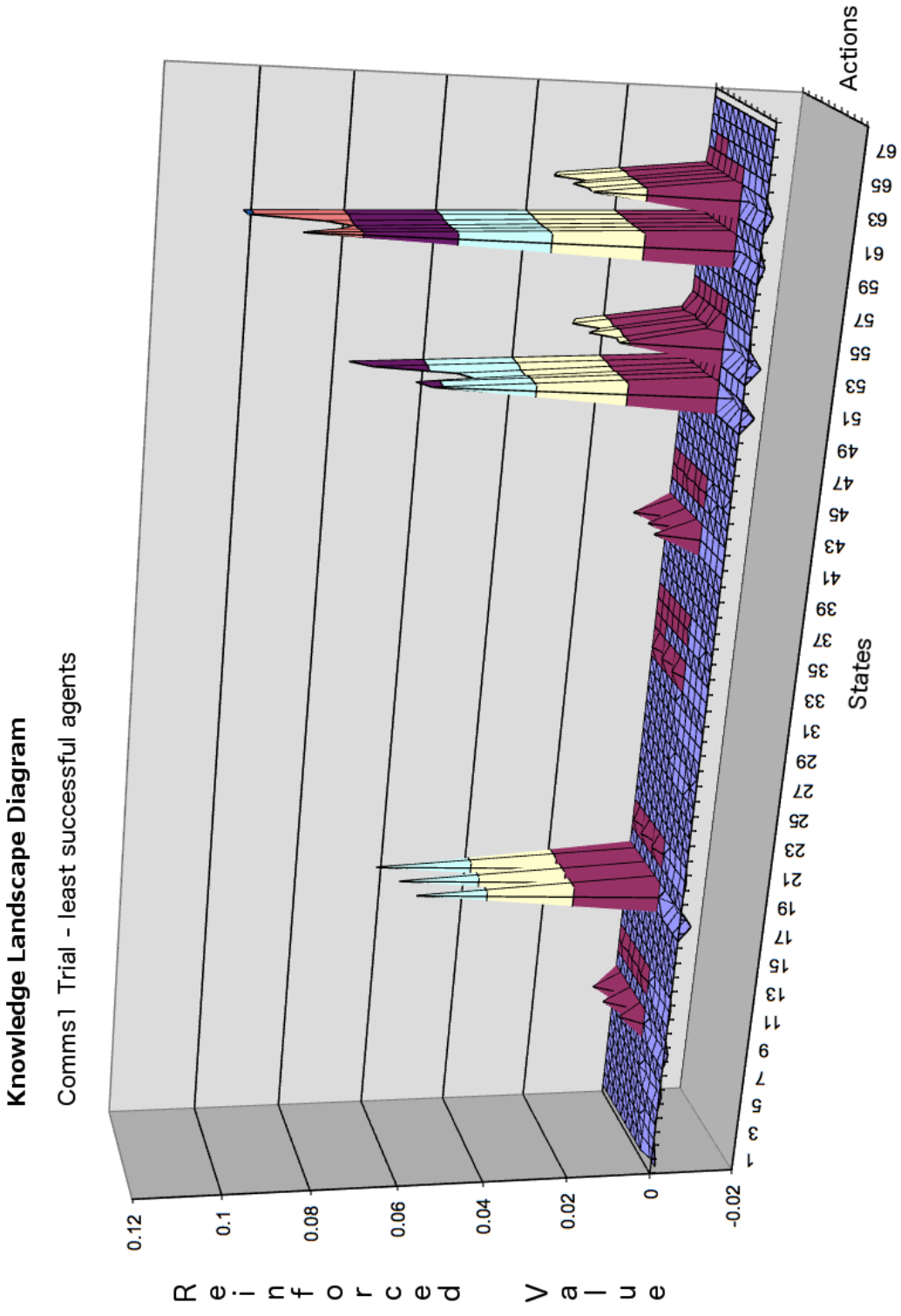


*Figure 5.21: Comms1 trial knowledge landscape diagram - least successful agents*

The knowledge diagram for the least successful agents shown above is marked by very little negative reinforcement. The negative reinforcement that is evident comes

from the agents own learning experiences. Again, this compares with the reinforcement of actions shown in the knowledge landscapes for the *comms3* trial in Figure 5.19 and Figure 5.20. As we would expect, the peaks in the positive reinforcement are more pronounced in places than the counterparts for the *comms3* trial. The corresponding knowledge landscape diagram for the most successful agents in the *comms1* trial is shown in Figure 5.22. As we would expect from the previous discussions, this diagram displays similar knowledge as that shown in Figure 5.21, just far more consolidated with stronger peaks emphasizing the further positive reinforcement of the existing peaks in Figure 5.21.

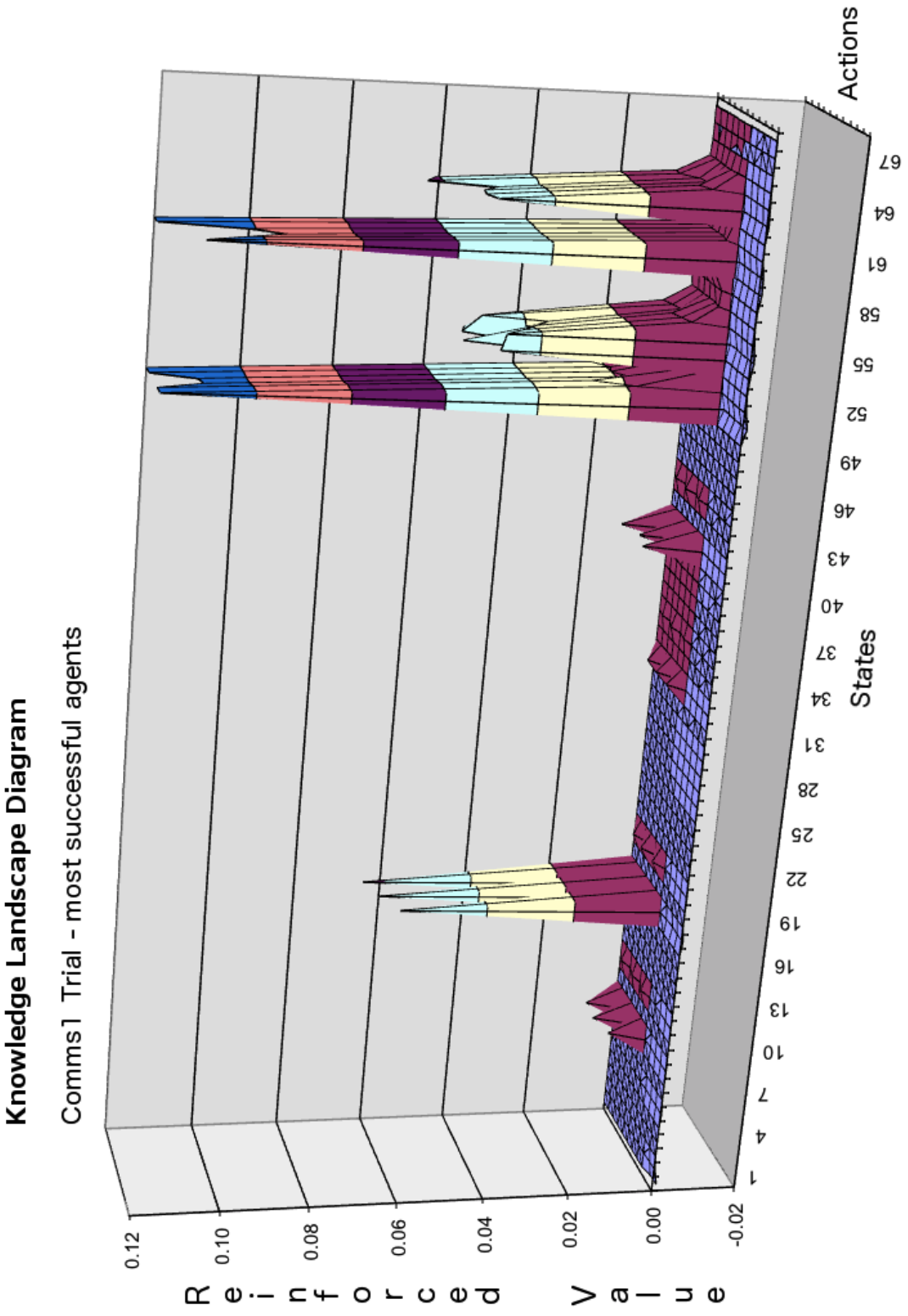


*Figure 5.22: Comms1 trial knowledge landscape diagram - most successful agents*

While the results from the *comms1* trial indicates that communication positive

learning events does not yield as strong a performance by the group as when they communicate all learning events, the communication still results in a stronger performance by the group than when they behave as a group of individual learners. In the *comms1* trial the reinforcement learning agents succeed in approximately 48 percent of simulation runs, whereas in the *benchmark* they succeed in only 24 percent of simulation runs, see Figure 5.10.

### 5.9.4 Comms 2 Trial

To contrast against the *comms1* trial, in the *comms2* trial agents were able to communicate, but only communicate the learning events that resulted in a negative reward. The knowledge landscape diagram for the least successful agents in the *comms2* trial is shown in Figure 5.23.

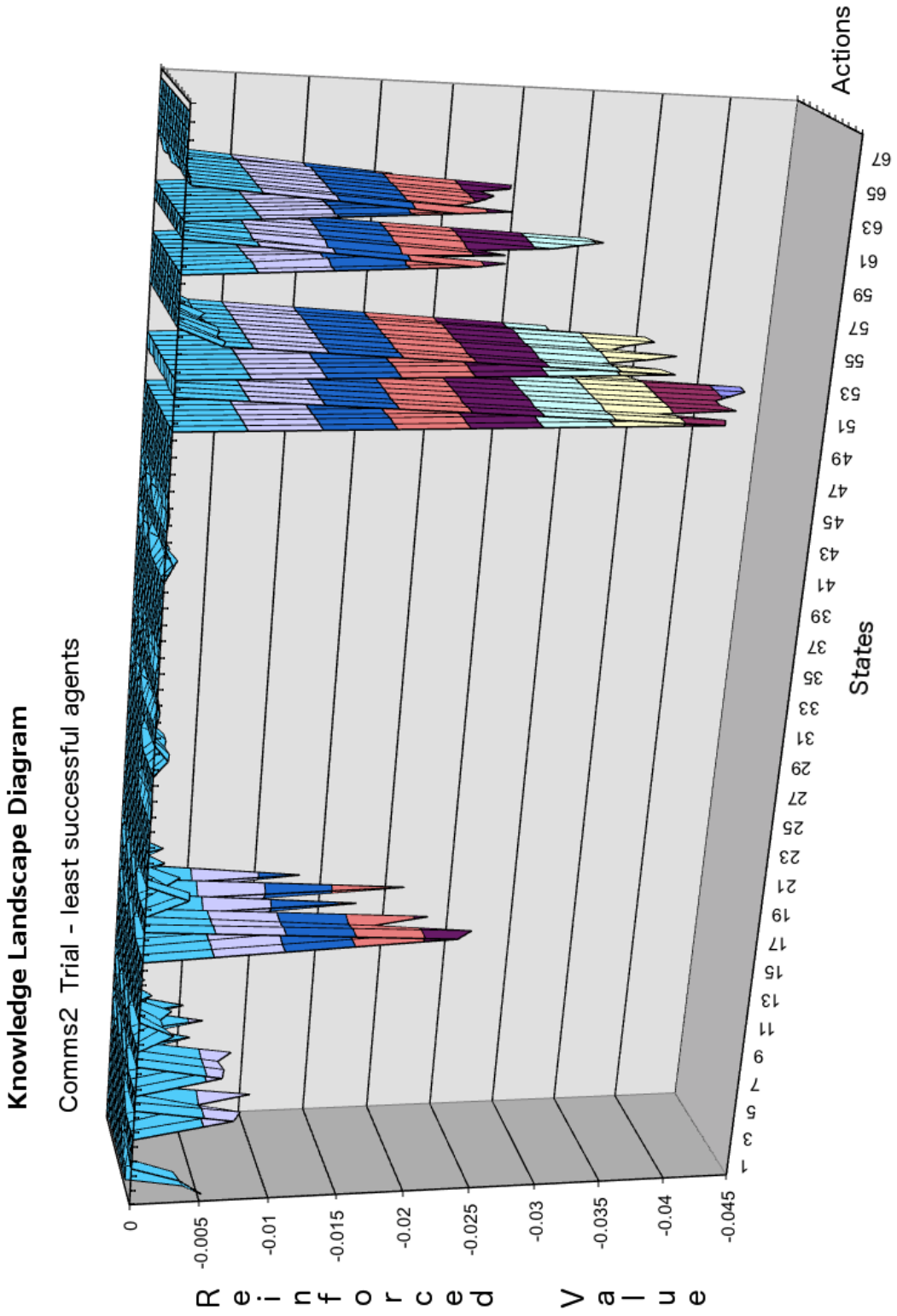


*Figure 5.23: Comms2 trial knowledge landscape diagram - least successful agents*

The knowledge landscape diagram for least successful agents in then *comms2* trial

shown in Figure 5.23 shows the result continual negative reinforcement from the communicated learning events. Interestingly, no positive reinforcement is observable in the diagram, even due to the agent's own learning events. There is a series of strong negatively reinforced peaks in states 50 – 64, including those actions (6 – 11) that have been strongly reinforced through positive learning events in the other knowledge landscapes. It should be noted that the knowledge landscape diagram in Figure 5.23 is not scaled the same as the other diagrams presented previously. The scale is approximately double that of the previous diagrams so that the extent of the negative reinforcement of the previously positively reinforced actions in the states 50 – 64 could be viewed obviously.

The knowledge landscape diagram for the most successful agent in the *comms2* trial can be viewed in Figure 5.24. This diagram is scaled correctly in accord with the other landscape diagrams previously presented.

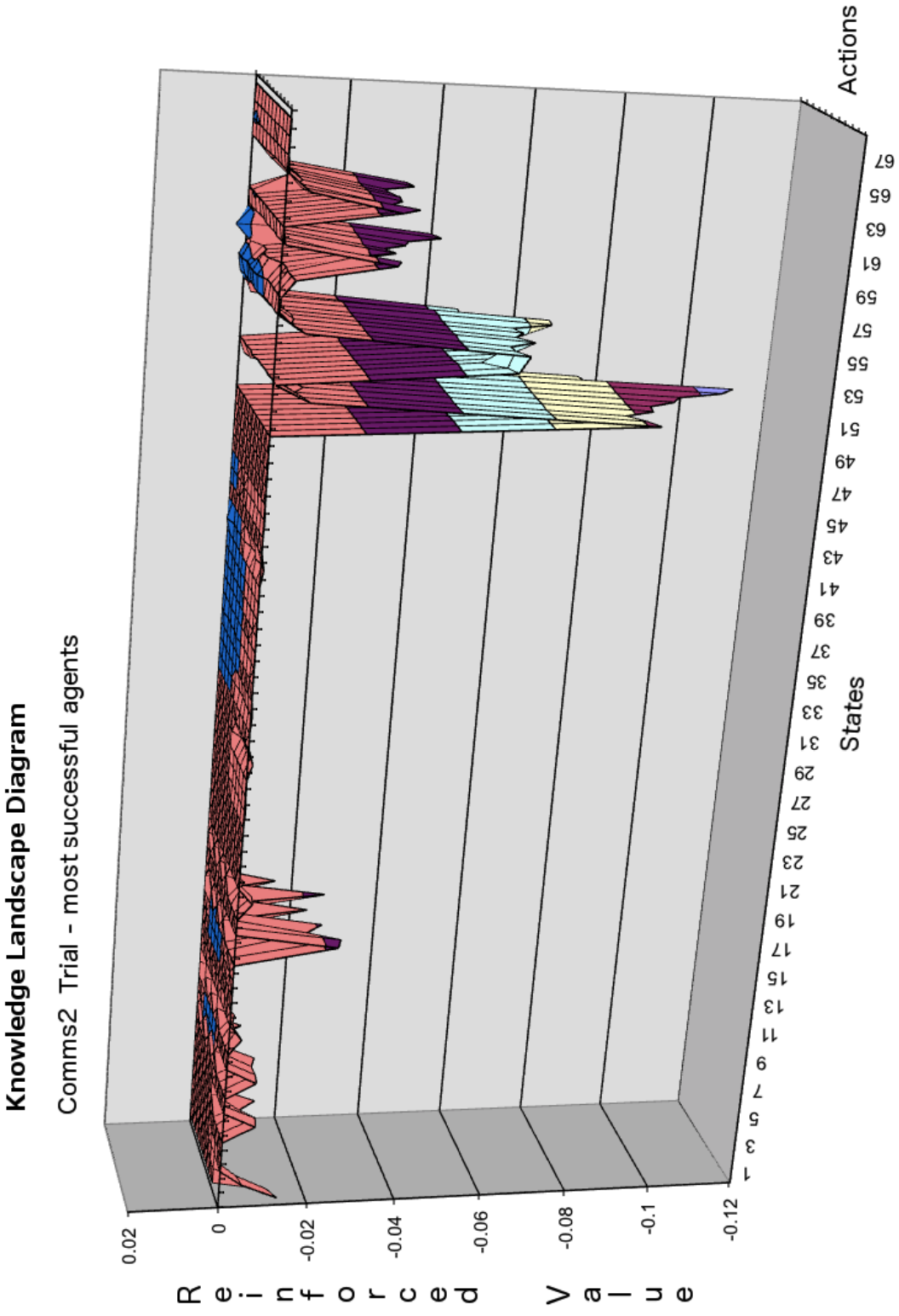


*Figure 5.24: Comms2 trial knowledge landscape diagram - most successful agents*

The knowledge landscape diagram in Figure 5.24 does show some minor positive reinforcement for some actions in states 31 – 43, and state 58, but this is due solely to

its own learning experiences. The communication of the learning events resulting from a negative reward leads to a performance drop for the reinforcement learning agents. For the *comms2* trial they only succeed in 16 percent of the cases compared to approximately 24 percent of simulation runs for the *benchmark.* These results will be discussed in the following section.

## 5.10 Discussion

By allowing agents in a Q-learning team to communicate their rewards, the final outcome is a significant increase in the Q-learning team's chance of success. The increase in performance changes from 24 percent approximately in the *benchmark* trial to approximately 52 percent in the *comms3* trial, see Figure 5.10. We determined in a previous section that this significant increase in the success rate is due to the communication ability of the agents as they communicate their learning events to other members of the reinforcement learning team. Initial investigation of various simulation metrics (Darbyshire and Wang, 2003), revealed little or no difference, thus a further line of inquiry would be to investigate changes within the agents themselves. The communication is explicit and takes place in the simulation as the agents experience learning events as the simulation unfolds. The end result of the communication is significant so the result of the communication on the group of agents $X=\{x_1, x_2, \ldots x_n\}$ must be discernible, and in a previous section we indicated this by $\Delta_{t+1} X$, which is the discernible effect on the group at time *t+1* due to $M(t)$, the communication of the group at time *t*. The changes in $X$ from time $t$ to *t+1*, $\Delta_{t+1} X$, will be realized in the individual agents as $\forall x_i \in X$, $\Delta_{t+1} X=\{\delta_{t+1} x_i\}$. In an agent utilizing a Q-learning, learning algorithm, any changes in the agent would manifest themselves in the agent's state-action matrix. Thus the state-action matrix becomes the focus to search for any change $\Delta_{t+1} X$.

The state-action matrix represents the agent's accumulated knowledge in choosing actions based on the current state. Thus, it represents the agents learned behavior. The knowledge landscape diagrams presented in the previous sections visually demonstrate the effect on the agents learned behavior from the communication which occurs during

the simulation trials. The differences between Figure 5.18 and Figure 5.20 in the previous sections depicts this effect by contrasting the learned behavior of the most successful agents in the *benchmark* trial with that of the most successful agents in the *comms3* run. There is a marked difference in the consolidation and reinforcement of rewards, both positive and negative for various combinations of agent states and actions. Communication has allowed these agents to effectively experience many more learning events, the result which is the consolidation of the rewards for those state-action values in the agents state-action matrix.

The differences between Figure 5.18 and Figure 5.20, can further be viewed in the associated contour diagrams for these figures. The contour diagrams represent an overhead view of the knowledge landscape diagrams to enable easier identification of specific states and actions of interest, and should be viewed together with the corresponding knowledge landscape diagrams. The contour diagrams for Figure 5.18, the knowledge landscape diagram for the most successful agents in the *benchmark* trial is shown below in Figure 5.25.

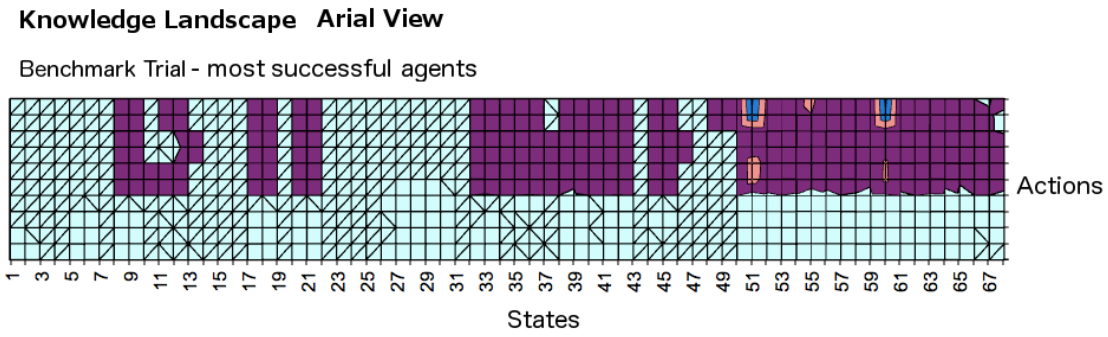


*Figure 5.25: Benchmark trial contour diagram - most successful agents*

The contour diagram in Figure 5.25 gives a good perspective on the knowledge landscape diagram for determining precise points of reinforcement, while the knowledge landscape diagram provides visual evidence of the degree of reinforcement for various state-action combinations. The contour of the knowledge landscape diagram representing the most successful agents in the *comms3* trial is shown below in Figure 5.26.

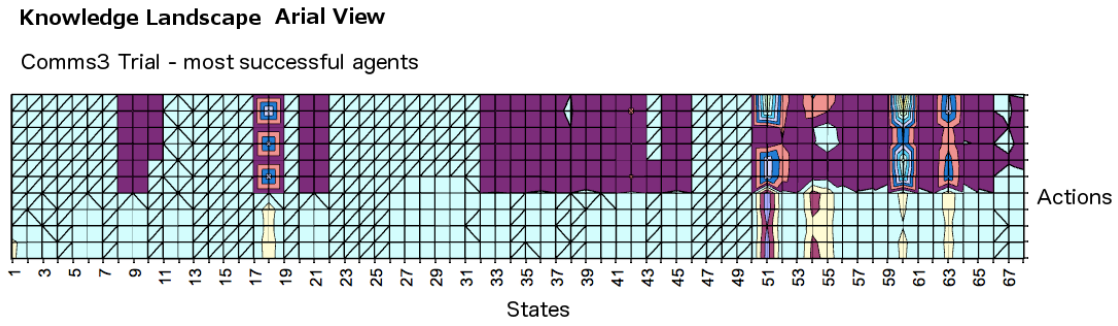


*Figure 5.26: Comms3 trial contour diagram - most successful agents*

Examining these two contour diagrams leads to some interesting questions. In Q-learning, when an agent identifies its current state, the action selection algorithm selects an action based on the probability of obtaining the greatest possible reward. This of course is apart from those instances when the action selection algorithm selects an action by random exploration. Thus, the action chosen will correspond to the element in the state-action matrix for the current state which has the maximum value. If we further compare the two contour diagrams for the most successful agents for the *benchmark* and *comms3* runs. While the agents in the *comms3* run have a far greater degree of reward consolidation, the agents in the benchmark run still have positive values recorded in their state-action matrix for a similar range of states and actions. This can be observed in the knowledge landscape diagrams in Figure 5.18 and Figure 5.20, but is more evident in the contour diagrams in Figure 5.25 and Figure 5.26

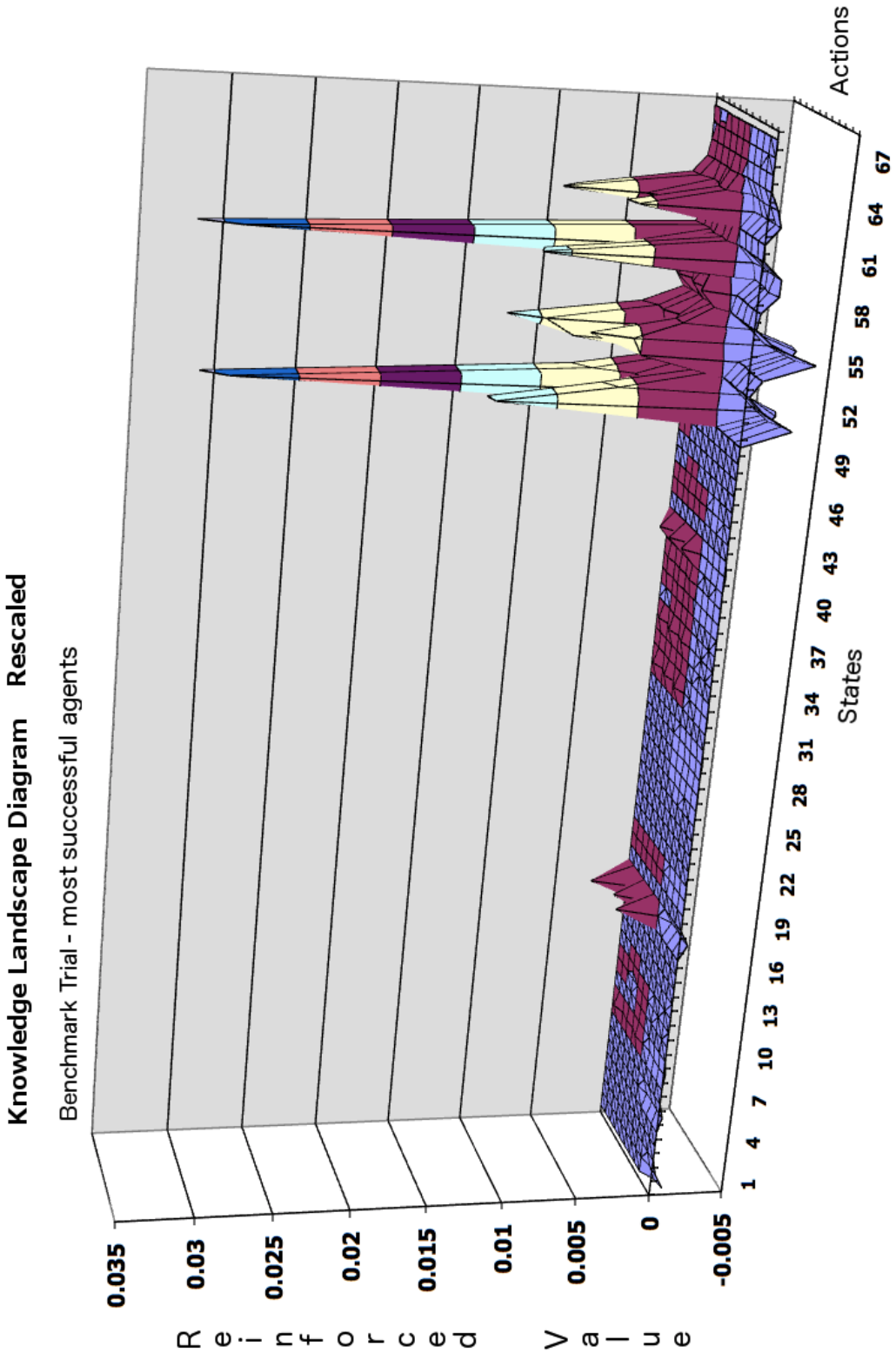


*Figure 5.27: Benchmark trial knowledge landscape diagram - most successful agents Rescaled*

Thus it seems that the agents in the *benchmark* trial and the agents in the *comms3*

trial are learning that similar actions are successful in the same states, yet the agents in the *comms3* trial are far more successful. To highlight the similarities even further Figure 5.27 shows the knowledge landscape diagram for the most successful agents in the *benchmark* trial, again, but rescaled by a factor of approximately three. If we compare this diagram in Figure 5.27 to the knowledge landscape diagram of the most successful agents in the *comm3* trial in Figure 5.20, the similarities are striking.

If the agents in the two trials are learning the same actions, then we must question why the agents in the *comms3* trial are much more successful? The difference between the two trials is of course $\{M(t)\}_{t=1.....T}$, the communication of the group at time *t* as *t* goes from *1* to *T,* the last simulation time-step. At the individual agent level, for agent $x_i \in X$, $M_i(t)$ is the set of messages received by agent $x_i$ at time t. Thus at the individual agent level, then differences between the *benchmark* and *comms3* trials is $\{M_i(t)\}_{t=1...T}$. From *Section 5.7.5 Initial Discussion*, we know that $M(t)$ increases the learning rate of the group $l(X)$, at time *t*, see Figure 5.12, thus $M_i(t)$ must increase the learning rate of the agent $x_i \in X$. By increasing the learning rate of an agent it will allow the agent to quickly consolidate the knowledge gained by other agents into its state-action matrix.

We can see this by examining the knowledge landscapes diagrams of the most successful agents in the *comms3* trial and least successful agents in the *comms3* trial. In *Section 5.9.2 Comms3 Trial* we noted the similarities between the diagrams apart from some further reinforcement both negative and positive on some actions for the most successful agents. This is to be expected since the most successful agents remain alive in the simulation to receive learning events with which to apply reinforcement using Equation (5.2). To confirm the similarities we can further examine the state-action matrices of two specific agents. In *section 5.9.2 Comms3 Trial* we chose one of the least successful agents from a specific simulation run, run 203, and one of the most successful agents from the same run. The contents of the state-action matrix for agent 26 in simulation run 203 can be seen in Listing 5.6 and Listing 5.7, this represents the least successful agent in the simulation run. The most successful agent in the simulation run is agent 15, and its state-action matrix can be seen in Listing 5.8 and Listing 5.9. As

observed in *section 5.9.2 Comms3 Trial,* the similarities are even more acute when viewing the state-action matrices of agent 26 and agent 15. It could be possible that the least successful agent in the comms3 trial is alive longer than its counterpart in the benchmark trial and is thus able to accumulate more learning events on its own. To counter this, the relevant output files from the simulation runs were scanned for further information.

Agent 26 in simulation run 203 of the *comms3* trial, the least successful agent in the simulation run died at simulation time 129, while the corresponding least successful agent in the benchmark trial whose state-action matrix is shown in Listing 5.2 and Listing 5.3 of *Section 5.9.1 Benchmark Trial*, died in simulation run of the benchmark trial at simulation time 121. Thus agent 26 of the *comms3* simulation 203 was only alive for 8 further simulation time steps. However the state-action matrix in Listing 5.6 and Listing 5.7 showed a level of maturity in terms of action reinforcement comparable to agents still alive at the end of the simulation run,

Thus, the early consolidation of rewards due to agents receiving early learning events from other agents, helps to quickly elevate the learned behavior of all the agents (including the first 5 to die) to more mature levels. The agents continue to further consolidate rewards from the environment, but the initial elevation to mature levels provide the agents the knowledge they need to increase their survival chances.

The reason for this lies in the fact that the simulation is non-deterministic. Thus the same choice of actions for a given state will yield different reward results at different points throughout the simulation. The interaction with other agents, and their subsequent concurrent learning, cause fluctuations in the state transition probabilities. Thus choosing action $a_i$ in state $s_j$ at time $t$ may result in a positive reward, but at some future time $t$, may result in a negative reward. Thus, if the actions which lead to more positive rewards later in the simulation aren't positively rewarded enough early in the simulation, their use may be discouraged by the action selection algorithm due to their negative reinforcement in the state-action matrix. The elevation of the state-action matrix to more mature levels early in the simulation allows the dominant actions evident in the knowledge landscape diagrams for the *comms3* trial to be reinforced positively.

This is enough to allow agents to survive the fluctuations of the state transition probabilities throughout the simulation by choosing these actions as the simulation progresses.

The discernible changes then in the individual agents $\{\delta_t x_i\}_{t=1\ldots T}$ due to $\{M_i(t)\}_{t=1\ldots T}$ can be seen in the early stages of the simulation, that is when $t$ is relatively small. The communication $\{M_i(t)\}_{t=1\ldots T}$ leads to an increase in the learning rate for $x_i$, which results in the state-action matrix reaching more mature levels early in the simulation. Since the learning rate in the benchmark trial was low, these mature levels of the state-action matrix could not be reached early and in a hostile environment, the levels of knowledge accumulated about actions leading to positive rewards coupled with the fluctuation of the state transition probabilities was not enough for the agent to be able to choose success actions.

The problems with the *comms2* trial where agents were only allowed to communicate negative rewards again resulted from the fluctuations in the state transition probabilities. As only learning events resulting in negative rewards were transmitted, the state-action matrices were negatively reinforced with information about actions leading to negative rewards. However, due to the successful actions being moving targets, in the normal course of events, some of these actions would eventually lead to positive rewards. But the continual input of unsuccessful learning events would continue to negatively reinforce these actions. Given the learning rate increase, the agent's own positive experience would not be enough to positively reinforce these actions enough so that they would be chosen, except when the algorithm was choosing a random action due to exploration, Hence the *comms2* trial performed worse than the *benchmark*.

The *comms1* trial performed significantly better that the *benchmark*, but not as good as the *comms3* trial. This was again to be expected in hindsight. Communication of only positive learning events did positively reinforce the state-action matrices as can be seen in the knowledge landscape diagrams in Figure 5.21 and Figure 5.22. This enabled the agents to choose actions early that would lead to positive learning events. However, again due to the non-markov nature of the multi-agent simulation and the

fluctuations in the state transition probabilities, some of those actions would also lead to negative learning events as the simulation unfolded, but the continual positive reinforcement would stop those actions from receiving negative reinforcement. Thus the success of the agents due to the early learning events leading to positive rewards, while enough to cause the agents to do well, would plateau at some stage at a level somewhat less than in the *comms3* trial. In the *comms3* trial where learning events initiated from both positive and negative rewards were communicated, the agents can better adapt to the changing probabilities of the state transitions and successful actions.

# 6

# MODELING COMMUNICATION

*The single biggest problem in communication is the illusion that it has taken place.*

– *George Bernard Shaw*

We have seen from past research and from the results and analysis presented in the previous chapter, *Chapter 5 Simulation Results*, that communication between cooperating agents has been shown to enhance the learning process of agents and their ability to complete tasks successfully in a group environment. In this research the environment was that of an abstract battlefield modeled by a distillation where two competing groups of agents competed to eliminate the other group. This represents an extremely hostile environment for an agent, and in such cases the ability of the agent to learn and adapt to the environment is crucial to its survival. As was shown for this particular model in the previous chapter, the learning rate of the group of blue agents (representing a group of learning agents) was not sufficient to ensure the success, and hence the survival of the group.

When the group introduced communication and allowed the agents to communicate their learning experiences the group was able to learn at an accelerated rate and thus achieve success in the majority of cases. In this particular simulation model the communication was vital for ensuring the survival of learning agents, and hence the survival of the group. From Chapter 5, we analyzed and studied the effects of the communication on the learning group and so understand the dynamics, but for

further understanding we do need an effective way to model such communication.

As stated in Chapter 1 Introduction, the main aim of this research was to develop a model for cooperative learning and study the effects of the communication on a group of agents. By doing this, the underlying effects on the individual agents could be studied to help fill in gaps identified within the literature. However as an adjunct to this research, we look at developing a model to represent the communication in such a system as described by the research during the previous chapters. While this is not intended to be complete or a definitive model, we view this as a beginning to further research on this important topic.

## 6.1 Communication in a Complex Adaptive System

It's difficult to define a complex system, as many of the definitions concentrate on different facets that make a complex system what it is. Numerous definitions vaguely describe complexity as being midway between order and chaos, but a common thread running through many definitions is that a complex system is comprised of a large number of interacting components that interact in a nonlinear fashion. The battlefield environment described in previous chapters aptly suits this definition as it comprises many components, all adapting to the environment and interacting in a nonlinear manner. In constructing such a simulation, one of the more important facets is that of the agent communication during the simulation progression.

Interaction between agents is central to the design of a Multi-agent System (Ferber, 1999), and is a consequence of their plural nature. However, modeling the actual communication between agents in a complex adaptive system, is difficult, as the elements of the communication system could further be considered as a complex system itself. Within the literature, most references for modeling communication are mainly concerned with the structure of the communication language, the semantics of the messages passed or the communication structures (Nickles et al., 2004), (Ghavamzadeh and Mahadevan, 2004), (Skrzypczynski, 2004). Artificial Neural Networks provide a convenient tool to describe the communication taking place within a multi-agent system in a more rigorous fashion.

## 6.2 Communications Systems View

In trying to model the communication within the simulation described above, we take a communication systems view (Ferber, 1999) of the communication component. That is, the communications, not the environment or agents becomes the focus of attention of the model. This is depicted in FFigure 6.1.

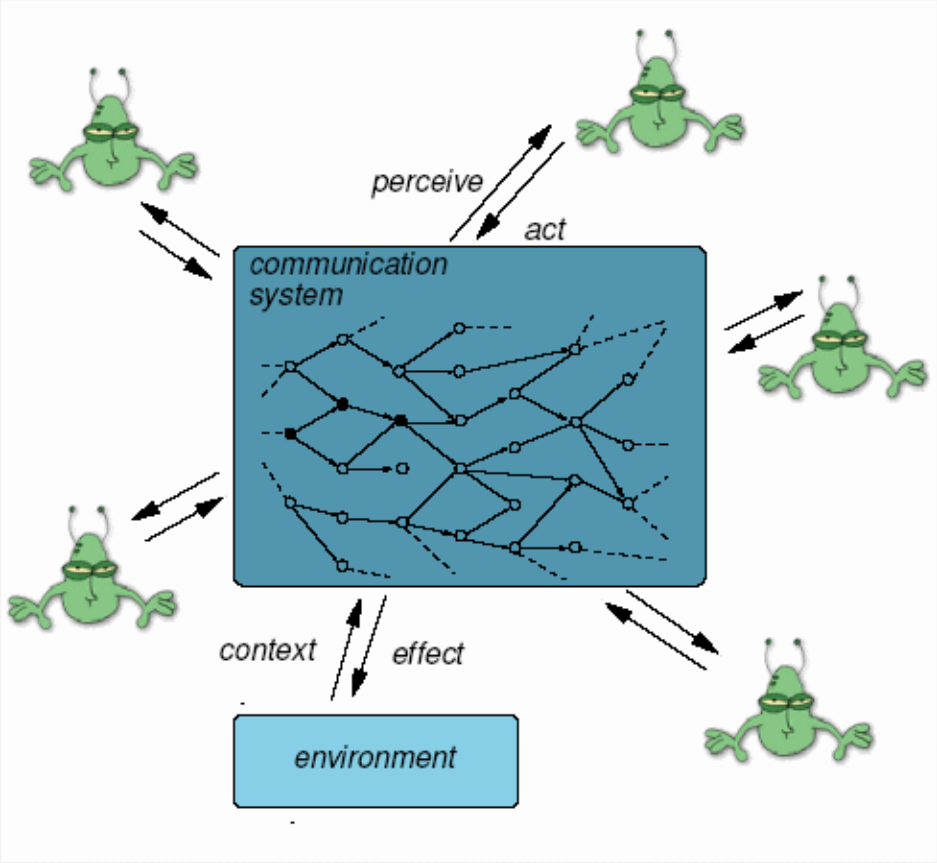


*Figure 6.1: Communications systems view (adapted from (Nickles et al., 2004))*

The relatively simple setup of the communication system described in Chapter 5 resembles that of a communication broadcast system. Thus all $n$ agents of the Q-learning team are effectively connected to all other agents within the same group, forming in effect, a fully connected communication network. The agents themselves are situated in their environment and can move around, thus effectively changing the shape of the graph, however, this can be ignored as we are only interested in modeling the

communication of the agents, thus the connectivity here is important, not the physical position of the vertices. Such a static communication system of $n$ agents could be modeled with a static graph $G = (V, E)$ where $|V(G)| = n$ and the set of edges, $E(G)$, would represent the communication channels (Green, 1993). In fact, the set-up as described could be represented by the complete graph $K_n$. Although the results of the communication in the experimental design indicated success, the current design itself is clearly not scalable for large values of $n$ if the simulation is to represent a realistic combat model.

In a complete graph on $n$ vertices ($K_n$), there are $n(n-1)/2$ edges, thus each agent could potentially be processing ($n$-$1$) communication messages at each distinct time period $t$ in the simulation. Clearly if we wish the simulation to remain realistic, such a system is not scalable when modeling for large values of $n$. To avoid this, the agents could use directed messaging to only smaller lists of specific agents, or more commonly use a signal propagation system similar to a broadcast but where the strength of the signal decreases over distance (Nickles et al., 2004). In the latter case, as one agent moves too far from another, the broadcast signal would get weaker, eventually fading altogether. Thus between two agents, $x_i$ and $x_j$, the strength of the message signal received by $x_i$, $V(x_i)$, is usually given by an equation of the form

$$V(x_i) = \frac{V(x_j)}{dist(x_i, x_j)}$$

or utilizing the square of the distance (Nickles et al., 2004). Implementing directed messages would be easier, but given the type of learning paradigm the agents are utilizing, the signal propagation method could be more suitable as the learning events taking place in an agents immediate neighborhood will be more relevant.

With this type of communication taking place, we can no longer model the communication as a static graph, since edges will appear and disappear as agents move in and out of range. We could however utilize a dynamic graph model. In this case we

model the communication channels with a dynamic graph $G = (V, E)$ where $|V(G)| = n$ and where at time $t$ . $E_t(G) \subseteq E(K_n)$ . Thus the communication channels are then represented by a series of timed edge changes $E_{t_0}, E_{t_0}, E_{t_0}, \ldots$ (O'Dell and Wattenhofer, 2005). The dynamic graph model is useful for studying the connectivity of the agents, but does not aid in the understanding of the communication that takes place between the agents. A more useful model to promote this understanding is that of an Artificial Neural Network (or simply a Neural Network).

Bossomaier (Bossomaier, 2000) discusses the issues of Complexity and the application of Neural Networks to many areas classed as complex where vast interconnectivity is featured. The communication methods utilized in *Chapter 5 Simulation Results* won't scale as the size of the learning agent group increases if we wish to remain true to the underlying domain. The complexity increases as the number of agents does giving rise to more interacting components. In such circumstances, a neural network description of the group of agents can be used effectively. Rather than utilize the neural net at a low level to mimic an individual agents learning capability, we can use the neural net at a more macro level where each node represents an agent. The communication in the group can then be modeled by the neural network utilizing the weighted input vectors to each node in the network.

## 6.3 Modeling Communication with a Neural Network

If we adopt a neural network approach to modeling, then each of the agents $X = \{x_1, x_2, \ldots x_n\}$ become nodes in the neural network. Each node has a number of weighted input vectors, corresponding to outputs from each of the other agents, as well as a feedback loop to itself. This in effect describes a recurrent neural network model, or more specifically a locally recurrent neural net. Figure 6.2 Shows a high-level overview of such a neural network model of a system with only 5 agents.

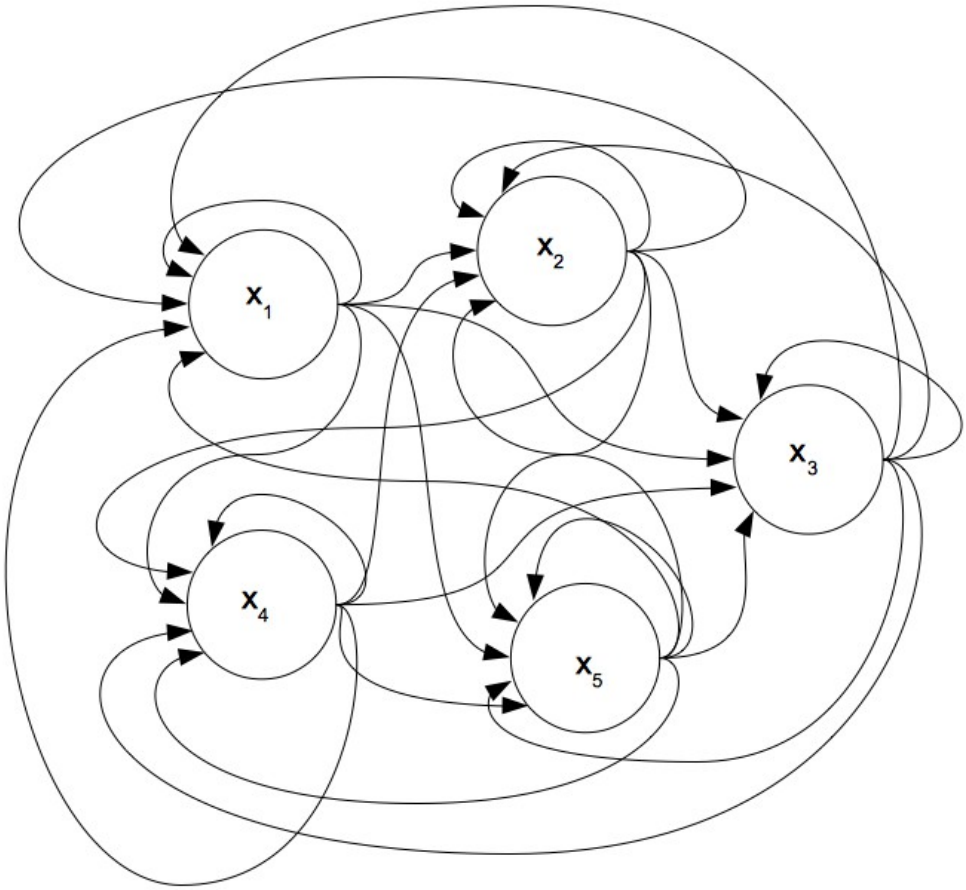


*Figure 6.2: Overview of neural network model with 5 nodes*

A more detailed diagram showing the detail of the feedback loop and weighted vectors for each node in the network is depicted in Figure 6.3. In order for the agent to learn, it perceives the state of its environment and initiates an action at time *t*. At time *t+1*, the agent then perceives its state again and possibly receives a reward for the action taken at time *t*. The learning matrix is then updated based on the original state at time *t* ( $s_t$ ), the action taken at time *t* ( $a_t$ ), the reward received at time *t+1* ( $r_{t+1}$ ) and the new state $s'_{t+1}$ at time *t+1*. This is represented by the 4-tuple $\langle s_t, a_t, r_{t+1}, s'_{t+1} \rangle$ on the feedback loop of the agent shown in Figure 6.3. This is enough information for the agent to update its learning matrix at time *t+1*. The feedback loop is normally represented internally in reinforcement learning, but can be represented as an external input vector in our model for completeness.

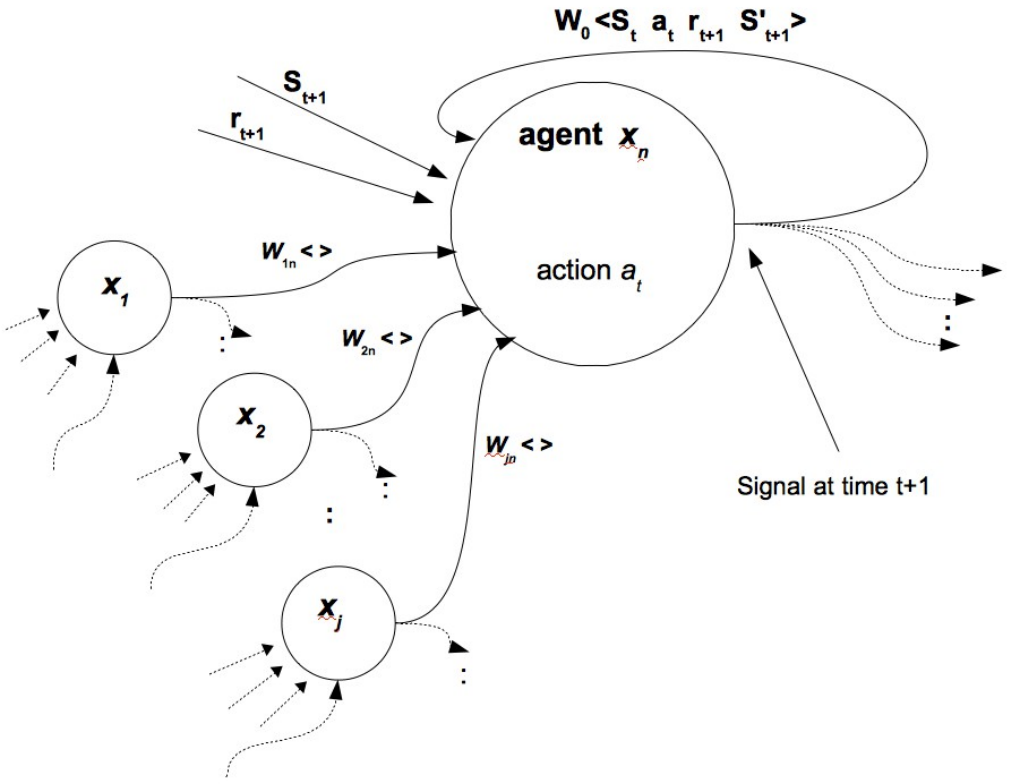


*Figure 6.3: Detail of nodes in neural net model*

On the weighted input vectors from agents $\{x_1, x_2, \ldots x_n\}$ shown in Figure 6.3, the symbol $\langle\ \rangle$ is used to indicate a 4-tuple of the form $\langle s_t, a_t, r_{t+1}, s'_{t+1}\rangle$ as previously described. The 4-tuples are sent by the different agents in order for the receiving agent to be able to effectuate multiple updates to the learning matrix. These form the input components of the weighted vectors and are the equivalent to the messages described in *Chapter 5 Simulation Results, Section 5.1 Messaging and Learning Events*. The weight component of a specific input vector to agent $x_i$, say $w_{ji}$, becomes Boolean-valued variable acting as a switch for the input vector

$$w_{ji} = \begin{cases} 1, & dist(x_j, x_i) \leqslant \Delta \\ 0, & dist(x_j, x_i) > \Delta \end{cases} \qquad j = 1, \ldots, n, \quad j \neq i \tag{6.1}$$

If the distance between the two agents is less than a specific quantity, $\Delta$, then the input vector is active, otherwise it is inactive. The fluctuation of $w_{ji}$ corresponds with

edge changes within the dynamic graph model. However, the distance between two agents isn't the only factor that determines whether an input vector is active. If agent $x_i$ does not experience a learning event at time *t*, then at time *t+1* the input vector from node $x_i$ to $x_j$ will not be active. Thus we can reformulate (6.1) as

$$w_{ji} = l_j \cdot d_{ji} \tag{6.2}$$

Where

$$\begin{aligned} l_j &= \begin{cases} 1, & x_j \text{ experiences a learning event at time } t \\ 0, & x_j \text{ does not experience a learning event at time } t \end{cases} \qquad j = 1,.. \\ d_{ji} &= \begin{cases} 1, & dist(x_j, x_i) \leqslant \Delta \\ 0, & dist(x_j, x_i) > \Delta \end{cases} \qquad j = 1,\ldots,n, \quad j \neq i \end{aligned} \tag{6.3}$$

I n *Chapter 5, Section 5.1 Messaging and Learning Events* and in *Chapter 1 Introduction*, a notation for the messaging between two agents and between all agents in a group was introduced. Specifically, a message between two agents $x_i$ and $x_j$ at time *t* is denoted by $m_{i,j}(t): x_i \rightarrow x_j$ (where each $m_{i,j}(t)$ is of the form $\langle s, a, r, s' \rangle$). The messages received by agent $x_j$ at time *t* is denoted by $M_j(t) = \{m_{i,j}(t)\}_{i=1\ldots n, i \neq j}$. The total communication of the set of agents $X = \{x_1, x_2, \ldots x_n\}$ at time *t* is then given by $M(t) = \{m_{i,j}(t)\}$.

In examining the input vector from say node $x_j$ to $x_n$ in Figure 6.3, then $w_{jn}$ is the switch to turn on or off message $m_{j,n}(t)$. Thus the combined action of the weights $\{w_{in}\}_{i=1\ldots j}$ in Figure 6.3 is to turn the input vectors on or off resulting in a dynamic $M_n(t)$ affected by learning the events of other agents and their distances from $x_n$.

Using the above notation, the neural network model in Figure 6.3 along with Equations (6.1) - (6.3) we can develop a model for the communication for a specific

node $x_j$ at time *t* as

$$M_j(t) = \left\{ \left[ l_i \cdot d_{ji} \right] m_{i,j}(t) \right\}_{i=1\ldots n, i \neq j}$$

whereas, the communication for the group of nodes $X = \{x_1, x_2, \ldots x_n\}$ at time *t* becomes

$$M(t) = \left\{ \left[ l_i \cdot d_{ji} \right] m_{i,j}(t) \right\}_{i=1\ldots n, i \neq j}^{j=1\ldots n}$$

The communication levels (the number of messages being passed) at time *t* for the above two models are

$$c(t) = \sum_{j=1\ldots n, j \neq i} \left[ l_j \cdot d_{ji} \right]$$

and

$$c(t) = \sum_{j=1\ldots n, j \neq i}^{i=1\ldots n} \left[ l_j \cdot d_{ji} \right]$$

respectively.

## 6.4 Defining the Objective Function

One of the crucial elements to the successful implementation of the reinforcement learning algorithm by a neural network is the development of the Objective function, or cost function. The objective function is used to train a neural-net and is used to provide a measure of system performance. Thus the objective function can be seen as an error function providing an error measurement between the actual output of the net, and the optimal output. We usually need to minimize this function. However, this implementation of a neural network is based on an underlying reinforcement learning

model, with the objective function governed by the underlying reward function. The reward function for the reinforcement learning algorithm was discussed at length in Chapter 5 Simulation Results, and is reproduced in Table 6.1 below.

*Table 6.1: Reward function for the simulation*

| | **reward value** | **action** |
|---|---|---|
| $r_t\ (s_i,a_i) =$ | 0.5 | damage to enemy |
| | -0.5 | receives damage |
| | 1.0 | kills enemy |
| | 0 | otherwise |

The reward function from Table 6.1 is governed by the simple algorithm

$$r = \begin{Bmatrix} 1 \\ 0.5 \\ -0.5 \\ 0 \end{Bmatrix}$$

depending whether the agent killed, damaged or was damaged by an enemy agent during the last time period. The reward function for an individual agent $x_i$ is returned as part of the signal $\langle s_{t-1}\ a_{t-1}\ r_t\ s'_t \rangle$, where the reward received $r_t$ at time *t* was for taking action $a_{t-1}$ at time *t-1* while in state *S*, which resulted in new state *S'* at time *t*. In this case, the object is to maximize the reward, as each reward results in an update to the agents state-action matrix, which in turn is used to decide which action to perform next based on selecting the action with the maximum state-action value.

Although the reward function is discrete, the signal produced by the reinforcement learning update function in *Chapter 5 Section 5.2 Model for Co-operative Learning*, is not. We can use the difference between the action value function $Q(s,a)$, the value of taking action *a* in state *s*, and $Q^*(s,a)$, the optimal action value function as the error differential. It is this difference we need to minimize. Thus the overall cost function of the network at time *t* can be denoted as

$$E(t) = \frac{1}{N}\sum_{i=1}^{N}(Q^{*}(s_i, a_i) - Q(s_i, a_i))$$

where *N* denotes the number of nodes (agents) in the neural network, and $Q(s_i, a_i)$ is the value of agent $x_i$ taking action $a_i$ in state $s_i$. For the purposes of the cost function each agent will only need concern itself with its own feedback input, and not the input from the other agents, as we can assume each other agent is also attempting to minimize its own cost function.

The difficulty with such a cost function from a practical viewpoint is that in a multi-agent simulation representing a complex system such as the one described, the optimal policy $V^{*}(s)$, and hence the optimal action value function $Q^{*}(s, a)$, are unknown. Additionally, given the complex nature of the system, they are not stationary as simulation time progresses. Thus the reward function offers the only way for each node in the network to pursue a minimization of the error, by attempting to maximize the reward.

# 7

# CONCLUSION

*A conclusion is simply the place where you got tired of thinking*

– *Dan Chaon (American Novelist)*

The objective of this research as outlined in *Chapter 1 Introduction,* was to investigate agent communication in a complex system amongst a group of agents co-operating to achieve a common goal. Previous research has shown that when such agents do cooperate they generally fare better as a team in achieving their goal. While the increase in performance has been documented, what was lacking in the literature was a detailed analysis of the cause-effect relationship between the communication and agent performance. If a group of agents $X=\{x_1, x_2, \ldots x_n\}$, performs better over time when they communicate, then there must be a discernible difference in the group at time *t*, say $\Delta_t X$ due to communication. Thus then, the central goal of the research is a through analysis of

$$\{\Delta_t X\}_{t=1\ldots T}$$

where *T* denotes the end of the simulation time. As *X* is composed of individual agents $x_1, x_2, \ldots x_n$, then the discernible change in the group due to communication $\Delta_t X$ must be realized in the individual agents. That is: $\forall x_i \in X$, $\Delta_t X=\{\delta_t x_i\}$. Thus in a through analysis of $\{\Delta_t X\}_{t=1\ldots T}$, we must analyze $\{\delta_t x_i \ \ \forall x_i \in X\}_{t=1\ldots T}$.

## 7.1 Summary

In order to conduct this research an appropriate simulation engine was needed. Initially, the CROCADILE Agent-based distillation engine developed at the Australian Defence Force Academy[1] (ADFA) (Barlow and Easton, 2002) was tested. This simulation engine had to be extensively modified and later re-written in order to accommodate intelligent agents utilizing reinforcement learning. The architecture of the simulation is a non-supervised multi-agent system representing a military distillation for studying simple land combat at a conceptual level. While the underlying domain of the distillation is a simple battlefield, other domains in which groups of agents compete to achieve a common goal could have been substituted. The advantage of the military distillation is that it provides us with a complex environment that is hostile to the agents, thus the effects of the communication on group success is of vital importance to the group. Details of the simulation tool developed for this research including the architecture and the theoretical aspects are covered in detail in *Chapter 4 Simulation Tool Design*. Sections of code from the simulation tool that are highlighted and discussed at length in Chapter 4, are listed in *Appendix A Simulation Code*. Not all code is provided due to space consideration, but only code specifically discussed in Chapter 4 in order to show important theoretical aspects of the simulation and the simulation architectural design.

In *Chapter 5 Simulation Results*, the fundamental objectives of the research are addressed. One of the important results of the research is the development of a model for cooperative learning in a multi-agent system. The model is developed in *Section 5.2 Model for co-operative Learning*, and extends a Q-Learning algorithm initially developed by Watkins (Watkins, 1989) and presented in (Sutton and Barto, 1998). Q-learning, and indeed reinforcement learning is used to allow a single agent to learn from its actions in an environment as it receives a reward from that environment. Developing a model for communicating in a multi-agent system utilizing reinforcement learning is challenging and any such model should be developed with regards to consistency with the underlying domain. In multi-agent reinforcement learning simulations, most shared learning is achieved with episodic learning, where agents swap learning experiences at

1 The Australian Defence Force Academy is a partnership between the Australian Defence Force (ADF) and the University of New South Wales (UNSW)

the end of the simulation run in order to prepare for the next. However, in a hostile environment where agents can be killed if they do not learn fast enough, this is unrealistic. Hence a temporal-difference learning algorithm was adopted as the basis for the new model so the agents could learn as the simulation progressed over time.

With the new model of cooperative learning, the learning rates for the group of agents *X*, is then bounded by the equation, $l(X) = n^2$, instead of the simple linear equation $l(X) = n$, where $l(X)$ represents the learning rate of the group of agents *X*, and the learning rate is measured by the number of learning events per simulation time step. See *Section 5.2 Model for Co-operative Learning*.

In *Section 5.3 Simulation Description*, the simulation parameters and set-up are described in detail. Included in this section is the development of the State Space and the reward function for the reinforcement learning agents, *Section 5.3.1 Initial State Space* and *Section 5.3.2 Reward Function*. Both of these are critical to reinforcement learning, as the state-space partitions the agents view of the world based on what is important in order for it to maximize its reward. Additionally the reward function must be designed to drive the learning of the agent. In a multi-agent system, the reward function allows a group to co-operate implicitly.

An initial *Benchmark* simulation trial is detailed in *Section 5.5 Initial Benchmark Results*. This simulation trial was designed to provide a benchmark for the simulation, pitting a untrained team of reinforcement learning agents against an experienced team. The reinforcement learning agents were individual learners, and the purpose of this trial was to provide a benchmark against which later simulation trials utilizing communication between the agents could be used to help isolate the results of the communication. Detailed results are shown in Section 5.5, but a decisive victory for the reinforcement learning agents occurred in only 3.2% of the simulation runs in the trial. Following on from this initial trial, the State Space was redefined along with the set of allowed actions for each agent in each state. The refinement divided the agents state space into 68 distinct possible states.

*Section 5.7 Experimental Trials*, describes the four fundamental trials developed in this research and used to analyze the effects of communication, that is, $\{\Delta_t X\}_{t=1\ldots T}$ and hence $\{\delta_t x_i \ \forall x_i \in X\}_{t=1\ldots T}$ discussed earlier. One of these trials is a new benchmark trial with the new state space to again provide the basis for comparison of the other trials. The four trials used for the analysis of communications include

- *Benchmark* – No communication, used a basis for comparison
- *Comms1* – Communication of positive learning events only
- *Comms2* – Communication of negative learning events only
- *Comms3* – Communication of all all learning events

A negative learning event is one where a agent undertook a specific action and received a negative reward from the environment, whereas with a positive learning event, the agent received a positive reward from the environment. See Section 5.7. In contrast to the original benchmark test with a success rate of 3.2%, the new benchmark recorded a success rate for the reinforcement learning agents of 24.2%. This marked difference came about as a result of the refinement to the reinforcement learning state space. These initial results and discussion are presented in *Section 5.75 Discussion,* along with a discussion of the comparative learning rates due to communication.

The success rates for the reinforcement learning agents in the different trials are:

- *Benchmark* – 24.2 percent
- *Comms1* – 48.4 percent
- *Comms2* – 16.2 percent
- *Comms3* – 52.1 percent

with the conclusion that the effects of communication were to enhance the performance of the reinforcement learning team significantly. This observation is consistent with previous research and provides evidence that the communication must have a discernible effect on the group of agents $X = \{x_1, x_2, \ldots x_n\}$. Before a detailed analysis of these effects on the agents was undertaken, a number of simulation metrics were investigated to ensure that there was indeed a discernible effect on $X = \{x_1, x_2, \ldots x_n\}$, and that the result was not an artifact of the simulation itself. This was investigated in *Section 5.8 Investigating Simulation Artifacts*.

A detailed analysis of the effects of communication on the reinforcement learning agents is undertaken in *Section 5.9 Examining the Knowledge Landscapes*. In this section, the agents are examined in detail to observe the effects communication has on them. In order to do this the state-action matrix of individual agents were examined in detail. The state-action matrix of a reinforcement learning agent is discussed in *Chapter 4 Simulation Tool Design*, and specifically in *Section 4.5 The Thinking Cycle*. The state-action matrix is essentially a two-dimensional array used to store double precision floating point numbers representing the stored reward values accumulated for performing various actions in specific states. The accumulation of the values is governed by the new model for reinforcement learning developed for communication in a multi-agent system, shown in *Equation 5.2* in *Section 5.2 Model for Co-operative Learning*.

By examining an agent's state-action matrix we can see exactly what an agent has learned. To aid in this detailed examination, the research constructed contour diagrams by plotting the values in the state-action matrices over the two-dimensional state-action matrix itself. This allowed us to observe the *Knowledge Landscape* of a particular agent. In order to understand the effects of the communication as a whole on the group of agents $X = \{x_1, x_2, \ldots x_n\}$, comparisons were performed on the most successful agents in $X$ with the least successful agents in $X$. By comparing the knowledge landscapes we can visually see the discernible effects of communication between these two sub-groups of $X$. An examination of the state-action matrix itself can then further pinpoint the

discernible changes in $\{\delta_t x_i \quad \forall x_i \in X\}_{t=1\ldots T}$ .

When comparing the knowledge landscapes between the least successful agents and the most successful agents in the communication trials, there was a surprisingly unexpected result. It was initially postulated that the communication would allow agents to learn 'different things'. That is, by allowing communication, each agent might expand its range of reward producing actions by incorporating learning events of the other agents which may have successfully applied different actions in different states. However it was discovered that the different groups of agents were essentially learning that the same actions were producing positive rewards. The similarities were more striking when we re-scaled the knowledge landscape diagrams of the least successful agents. Thus we could conclude that, rather than learning new things, the effect of the communication on the group of agents $X = \{x_1, x_2, \ldots x_n\}$ , was to positively reinforce the state-action matrices of the individual agents $x_1, x_2, \ldots x_n$ much earlier in the simulation than was possible when the agents were individual learners and did not share learning experiences. Intuitively, this corresponds to what we understand about the underlying domain of the simulation, where trained groups of combatants are equipped with communication devices and information is considered vital to mission success.

## 7.2 Critical Appraisal

As with all research, a number of questions always arise which require a critical evaluation of what has been done.

Being a new simulation, it is always difficult to judge the accuracy of its functioning without a basis for comparison. During the course of this research, a literature review did not reveal any military distillations utilizing reinforcement learning. However, the initial software, CROCADILE (Barlow and Easton, 2002), which the simulation tool used as a basis was developed at the Australian Defence Force Academy and had been used to study a number of simulation scenarios. The modifications and eventual re-write of the simulation utilized architectural concepts from CROCADILE and an Object Oriented approach to implement the simulation tool. Additionally, the original simulation design was modified to correct for possible

anomalies not taken into account in the original design. Specifically, the array of actions to be processed after each agent's thinking cycle was complete was randomized to correct for any bias due to the order of the agents in the cycle. Additionally, the use of Java threads was removed due to possible Operating System bias in thread processing (Darbyshire, 1998).

As a two point cross check, initial trials utilizing CROCADILE behaved analogously to the new simulation tool, ARES, see *Chapter 4 Simulation Tool Design*. Additionally, once communication was enabled between agents of the reinforcement learning group, the outcome was as expected and in line with previous research in so much as the agent's performance improved. It is difficult to gauge the performance levels in comparison with other research due to the nature of the reinforcement learning model utilized and the essential hostile nature of the environment for these agents. However, the performance gains due to communication indicate that the simulation was performing within reasonable bounds of expectation, given the differences.

The choice of Java as the language for implementation of the simulation tool could be questioned. Java is not as efficient as other languages such as C for use when real-time behavior is expected. However, this was a distillation, and as such, real-time behavior is not needed. Java did provide a platform independent implementation which was more important, and the Object Oriented nature of the language suited the treatment of agents as independent entities. The simulation tool did achieve its objectives in providing a platform in which to model a hostile complex system where agents could communicate in order to achieve group objectives.

The use of reinforcement learning for a multi-agent system requires some explanation. Reinforcement learning was initially designed around a state-space conforming to a Markov assumption, and thus limited to a single agent. A multi-agent system, while generally Markov-like in nature is essentially a non-Markov decision process, as the interaction between the agents themselves and concurrent learning affect the state transition probabilities (Arai et al., 1999). One of the measurements of performance often used for reinforcement learning algorithms is convergence, but in multi-agent reinforcement learning, convergence is not guaranteed. The main problem

stems from the non-stationary nature of the task due to hidden variables caused by other agents affecting the environment. Not withstanding these concerns many researchers have reported success in utilizing multi-agent reinforcement learning in simulations to study a wide range of team-based problems (Kelly and Keating, 1998, Mataric, 1998, Nunes and Oliveira, 2003, Preux et al., 2004, Sen et al., 1994, Tan, 1993). As a consequence, the simulation tool developed for this research has provided some valuable insights to the effects of communication on learning agents and provided avenues for further research discussed in S*ection 7.4 Further Research.*

In a *Section 7.1 Summary* we discussed the initial selection of the agents s*tate space*. The initial benchmark trial resulted in a success rate of only 3.2% for the reinforcement learning team. After refinement of the state space resulting in an environment partitioned into 68 distinct states, the new benchmark trial resulted in a success rate of 24.2%. This improved success was due to the refinement of the state space alone. It is conceivable that further refinement of the state space could result in further improvement of performance. However, as discussed in *Chapter 2 Literature Review*, there is no algorithm for constructing a state space for a reinforcement learning problem. The task is still regarded as more an art form than a science and too much valuable time could be consumed on such refinements. Given the improvements in performance of the agents when communication was implemented we could still effectively analyze the discernible improvements in the agents due to communication. It is theorized that communication would still provide a similar level of improvement over any further state space refinement.

Each of the four fundamental trials developed for this research and executed by the simulation tool consisted of 2000 simulation runs. Each simulation run was allowed to run for 1000 discrete time steps, or until a victory was reached. Also, each simulation run consisted of two opposing teams, the reinforcement learning team, the subject of this research and an opposing team representing a highly trained combat force. It could be argued that more simulation runs should have been performed per trial. However, during the initial stage testing 5000 simulation runs were attempted, but this was reduced to 2000 for two reasons. The time for each simulation trial was prohibitive, and the data output was extremely large. Additionally, the multi-agent

simulation is a non-Markov decision process, and thus there is no convergence. It was decided that 2000 simulation runs provided enough data for analysis as more runs would only produce more non-convergent data. In the initial trial with 5000 runs, the trends were similar and the extra data files did not provide any further information.

It should be noted that a possible form of biasing might have been introduced by the use of a weapon with unlimited ammunition. With unlimited ammunition, agents might overtime reinforce actions where they fire the weapon, leading to a biased reinforcement of these actions. This could produce a bias against negative reinforcement which might not occur if the ammunition was limited. Some of the actions which need to be negatively reinforced to aid in the agents survival could be outnumbered by less effective ones where firing is involved. We did note that the state-action matrices seemed to be sparsely populated and this needs to be investigated in any future research.

We believe that the simulation tool developed and the design of the research experiments, have helped overcome any theoretical concerns of the application of reinforcement learning to multi-agent systems. Many other researchers have applied similar techniques to their research and this has proved successful. We believe that this research has been successful in answering the fundamental research question of the effects of communication on performance of a group of agents in a complex system. Additionally, the development of the model for communication in a multi-agent system using reinforcement temporal-difference learning techniques has been valuable in providing a framework for similar simulation problems. Many insights and much experience has been gained, and the conference papers and Journal articles resulting from this research has generated interest. During this research some answers have been found, but also more questions posed. Some of these questions are outlined in *Section 7.4 Further Research*.

## 7.3 Contribution

As stated in *Chapter 1 Introduction*, this research makes a number of contributions to the field of study.

A new model of cooperative learning is developed through the implementation of an enhanced Q-learning update function which also includes learning events communicated by other agents. This is a modified version of the update function displayed in Sutton and Barto (Sutton and Barto, 1998). The original update function is for a single agent learning in an environment without the presence of other learning agents. The updated learning equation shown in *Section 5.2 Model for co-operative Learning*, was specifically developed for this research. This updated function is a model that allows not only for the reinforcement learning agent to update it's own learning events, but also allows further updates to the agent knowledge based on the communication input signals from the other agents in the group. This model has been tested in the simulation with good results showing enhanced learning by the group.

Further contributions also lay in the detailed analysis of the simulation runs. This analysis pinpoints the effects of the communication on the agents, and visualizes these using surface plot diagrams of the agents action-update matrix. These diagrams show how the group communications reinforce effective actions for the agents at an early stage in the simulation. However, as the multi-agent system is non-Markovian in nature, this becomes a moving target, and the surface diagrams evolve over time to reflect the changing nature of effective actions.

Finally, a Neural Network based model of the group communications is developed to effectively represent the communication in a more coherent form for further research. Such communication has always been difficult to represent at a micro level in a complex system, due to the non-linear nature of the systems. However, Neural Networks provide us with a good modeling tool to represent this information.

It should be acknowledged that the agents in this study communicate a very precise set of information that relates to the particular learning model and the internal

representation of the learning agents. The agents improve their learning in the domain represented by the simulation in this research. This particular method has not been evaluated beyond the domain in this study. While we hypothesize that the technique could be generalized, the research in this thesis is based on the domain discussed throughout the preceding chapters.

## 7.4 Further Research

Many opportunities for further investigation have presented themselves throughout the performance of this research but were beyond the scope of the current investigation. While this research has answered the fundamental question of where the effects of communication are realized in a multi-agent system, and what discernible changes this has on the individual agents, there is ample scope to extend this research to address further questions. In particular, the use of simulation to provide empirical evidence to help develop theory for the underlying domain. While Davis, Eisenhardt and Bingham provide a roadmap for helping to develop theory in the underlying domain through the use of simulations (Davis et al., 2007), other researchers suggest that many simulations are too abstract and strip away too much realism to provide any theoretical framework (Chattoe, 1998, Fine and Elsbach, 2000) (cited in (Davis et al., 2007)).

The simulation used in this research is a military distillation used to study land combat at a conceptual level. The addition of communication ability to the reinforcement learning agents resulted in improvement to the team performance, and additionally generated some interesting results in terms of initial accelerated learning rates and the reinforcement of early successful actions by the sharing of learning events. However, the type of communication implemented within the simulation was effectively a broadcast-type communication. When an agent experienced a learning event, it sent a message to all other agent in the same team, and while the messages were implemented individually, it was fundamentally a broadcast. Clearly such a system could not scale-up if we are trying to retain consistency with the underlying domain. The simulation in this research experimented with groups of 15 agents on each side, but if we added to this figure, then in the underlying domain we must reach a point where giving every agent effective access to a broadcast system becomes counter

productive. The number of messages for each agent to assimilate would become too high.

While in a discrete time simulation we can have an agent incorporate as many learning events as desired, in the underlying domain where action is progressing in real continuous time, this would be unrealistic. Additionally, if a large number of agents were spread out over a large geographical area in the simulation, the learning events in one locale might not be applicable to agents in a different location. This could reinforce the wrong actions needed to succeed. In the underlying domain this might be analogous to a large front forming on a battlefield and the required tactics may be different along various points of the conflict. Thus if we want the simulation to provide theoretical insights into the underlying domain a different communication mechanism need to be implemented.

One avenue for further research in this area is to investigate the model for communication developed in *Chapter 6 Modeling Communication,* utilizing a Neural Network model. In particular, if we consider the view of an individual agent as a node in a Neural Network of nodes sharing learning events, see Figure 6.3, then we can utilize the weights $w_1, w_2, \ldots w_n$ on the input vectors shown in Figure 6.3 by using Equations 6.1, 6.2 and 6.3. Thus the weight becomes a switch for the input vector based on distance from the receiving agent to the transmitting agent. So an agent will not process learning events from other agents where the distance between them is greater than a defined quantity $\Delta$. Research would be required to investigate the benefits of this approach and an optimal value for $\Delta$.

Another avenue for research utilizing the same Neural Network model would be to investigate the weights $w_1, w_2, \ldots w_n$ on the input vectors as a function of *trust*. Razaei (Rezaei, 2011) investigates issues of trust and reputation between agents for performance improvement and found strong supporting evidence through simulation experiments. Parsons, McBurney and Sklar (Parsons et al., 2010) discuss issues of trust between agents in a multi-agent system, and in particular the application argumentation to reason about trust. Argumentation would be an interesting approach for further research, utilizing the messages sent as the *support* of the argument and the reward

received by an agent using that message as a basis for an action, as the *conclusion* of the argument. By building trust relationships within the multi-agent simulation we can utilizing the weights as a measure of both trust and distance. This could help in creating a more realistic relationship between the simulation and the underlying domain and may allow us to take steps towards developing theory.

Two other areas for further investigation have emerged from this research. The issue of the *state space* was discussed in a previous section. A refinement of the state space of the reinforcement learning environment led to performance improvement even before the communication was enabled for the learning agents. Further investigation is needed into the relationships between this raw improvement and the state space. This is needed so that in future simulations, a set of guidelines may be followed to initially develop a state-space, rather than best guess followed by further refinement. Another related issue to the state space is that of the sparse population of the state-action matrix for the reinforcement learning agents. This was initially discussed in *Chapter 5 Simulation Results*. It may be that due to the hostile nature of the agent's environment and the relatively short time-line of the simulation runs before a decisive victory occurs, the agents do not get a chance to try all actions due to the need to maximize the reward function. It is also possible that the fundamental nature of the state space may play a part. In either case, answers to these questions will benefit the area of multi-agent simulation development utilizing reinforcement learning.

# BIBLIOGRAPHY

.

## References

ARAI, S., SYCARA, K. & PAYNE, T. 2000. Multi-agent reinforcement learning for Scheduling Multiple-Goals. Fourth International Conference on Multi-Agent Systems (ICMAS'2000). Boston, USA: IEEE Computer Society.

ARIMOTO, T. & UETA, S. 2009. Report on Emerging and Interdisciplinary Research Fields: Solving Social Issues and Expanding the Frontiers of Science and Technology. Center for Research and Development Strategy, Japan Science and Technology Agency. Document No. CRDS – FY2008 – XR – 02, http://crds.jst.go.jp/output/pdf/08xr02.pdf [Accessed 4/7/2012]

AXELROD, R. 2003. Advancing the Art of Simulation in the Social Sciences. *Japan Society for Management Information Systems (JASMIN),* Vol 12 No 3.

BALACHANDRAN, A., RABUYA, L. C., SHINDE, S. & TAKALKAR, A. 2000, Simulation and Modeling. Available: http://www.uh.edu/~lcr3600/simulation/entry.html [Accessed 28/2/2011].

BANKES, S. C. 2002. Agent-based modeling: a revolution? *Proceedings of the National Academy of Sciences,* Vol 99 No Suppl 3**,** pp7199-7200.

BARLOW, M. & EASTON, A. 2002. Crocodile - An Open Extensible Agent-based Distillation Engine. *Information and Security: An International Journal,* Vol 8, No. 1**,** pp17-51.

BECKERMAN, D. L. P. 1999. The Non-Linear Dynamics of War. Science Applications

International Corporation, ASSET Group. Available http://belisarius.com/modern_business_strategy/beckerman/non_linear.htm [Accessed 20/4/2004]

BERNDTSSON, M., HANSSON, J., OLSSON, B. & LUNDELL, B. 2002. *Planning and Implementing Your Final Year Project With Success,* London, Springer-Verlag.

BICKMAN, L. & ROG, D. J. 2008. Applied Research Design: A Practical Approach. *In:* BICKMAN, L. & ROG, D. J. (eds.) *Handbook of Applied Social Research Methods*. U.S.A.: SAGE Publications.

BONABEAU, E. 2002. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences,* Vol 99 No Suppl 3**,** pp 7280-7287.

BOSSOMAIER, T. 2000. Complexity and Neural Networks. In: BOSSOMAIER, T. & GREEN, D. (eds.) Complex Systems. United Kingdom: University Press.

BOSSOMAIER, T. & GREEN, D. 1999. Patterns in the Sand: Computers, Complexity, and Everyday Life, Perseus Publishing, ISBN:0738201723.

BOWLES, S. & GINTIS, H. 2003. The Origins of Human Cooperation. *In:* HAMMERSTEIN, P. (ed.) *The Genetic and Cultural Origins of Cooperation.* MIT Press.

BOWLING, M. & VELOSO, M. 2000. An Analysis of Stochastic Game Theory for Multiagent Reinforcement Learning. Computer Science Department, Carnegie Mellon University, PA, Report No CMU-CS-00-165

BOWLING, M. & VELOSO, M. 2002. Multiagent Learning Using a Variable learning Rate. *Artificial Intelligence,* Vol 136 No 2**,** pp 215-250.

BOYD, R. & RICHERSON, P. J. 2005. Solving the Puzzle of Human Cooperation. *In:* LEVINSON, S. (ed.) *Evolution and Culture.* Cambridge, MA: MIT Press.

BRANDSTEIN, A. Introduction to Project Albert (briefing slides). 4th Project Albert International Workshop, 2001 Cairns, Australia.

BRYSON, J., LOWE, W. & STEIN, L. A. 2001. Hypothesis Testing for Complex Agents. *National Institute of Standards and Technology Special Publication 970***,** pp 233-240. http://people.cbrss.harvard.edu/wlowe/Publications/nist.pdf [Accessed 12/5/2004]

CARPI, A. & EGGER, A. E. 2008. Research Methods: The Practice of Science. *Visionlearning* [Online], POS-2. Available: http://www.visionlearning.com/library/module_viewer.php?mid=148&l=&c3= [Accessed 4/3/2013].

CARVER, J., VAN VOORHIS, J. & BASILI, V. 2004. Understanding the Impact of Assumptions on Experimental Validity. International Symposium on Emperical Software Engineering, 2004 Redondo Beach, CA, USA. IEEE, pp 251-260.

CELLIER, E. F. & KOFMAN, E. 2006. *Continuous System Simulation,* New York, Springer, ISBN 0-387-26102-8

CHALKIADAKIS, G. 2003. Multiagent Reinforcement Learning: stochastic games with multiple learning players. Department of Computer Science, Univeristy of Toronto, Report March 2003

CHATTOE, E. 1998. Just how (un)realistic are evolutionary algorithms as representations of social processes? *Journal of Artificial Social Science Simulation,* Vol 1 No 3.

CHEUNG, V. & CANNONS, K. 2002. An Introduction to Neural Networks. Winnipeg, Manitoba, Canada: Signal & Data Processing Laboratory, University of Manitoba. Available http://www2.econ.iastate.edu/tesfatsi/NeuralNetworks.CheungCannonNotes.pdf, [Accessed 23/1/2012]

CLAUS, C. & BOUTILIER, C. 1998. The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems. Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, Madison, Winconsin. American Association for Artificial Intelligence, pp 746-752 ISBN 0-262-51098-7

COLTON, T. 2001. The Army Synthetic Environment. In Proceedings of SimTect, 2001, Canberra, Australia.

CONOLE, G., SCANLON, E., MUNDIN, P. & FARROW, R. 2010. Technology-enhanced learning as a site for interdisciplinary research. London: Technology Enhanced Learning Research Programme; Economic and Social Research Council; Institute of Education, http://www.tlrp.org/docs/TELInterdisciplinarity.pdf [Accessed 24/8/2012]

COULOM, R. 2002. Reinforcement Learning Using Neural Networks, with Application to Motor Control. PhD Thesis, Institut National Polytechnique de Grenoble, available [http://remi.coulom.free.fr/Thesis/], accessed June 2014

DARBYSHIRE, P. 1998. *An Investigation Into The Parallel Implementation Of JPEG For Image Compression.* Master of Engineering (research), Victoria University, Australia.

DARBYSHIRE, P. & WANG, D. 2003. Learning to Survive: Increased Learning Rates by Communication in a Multi-agent System. *In: AI 2003: Advances in Artificial Intelligence.* Heidelberg: Springer Berlin.

DAVIS, J. P., EISENHARDT, K. M. & BINGHAM, C. B. 2007. Developing Theory Through Simulation Methods. *Academy of Management Review,* Vol 32 No 2**,** 4pp 80-499.

DAWSON, C. W. 2005. *Projects in Computing and Information Systems: A Students Guide,* England, Pearson, ISBN 0-321-26355-3

DAYAN, P. & HINTON, G. E. 1993. Feudal Reinforcement Learning. *In:* HANSON, S. J., COWAN, J. D. & GILES, C. L., eds. Advances in Neural Information Processing Systems 5, 1993 Denver USA. Morgan Kaufmann, San Mateo, CA.

DOOLEY, K. 2002. Simulation Research Methods. *In:* BAUM, J. A. C. (ed.) *Companion to Organizations.* Wiley-Blackwell.

DORAN, J. E., FRANKLIN, S., JENNINGS, N. R. & NORMAN, T. J. 1997. On cooperation in multi-agent systems. *The Knowledge Engineering Review,* Vol 12 No 3**,** pp 309-314.

DÖRNYEI, Z. 1997. Psychological Processes in Cooperative Language Learning: Group Dynamics and Motivation. *The Modern Language Journal,* Vol 81 No 4**,** pp 482-493.

DRESHER, M. 1981. *The Mathematics of Games of Strategy,* Dover Publications, ISBN 978-0486642161

DUIVESTEIJN, W. 2006. Contimuous Simulation. Available: http://www.cs.uu.nl/docs/vakken/sim/continuous.pdf [Accessed 8/7/2011].

ENANORIA, W. 2013. Introduction to Survey Methodology. Available: http://www.idready.org/courses/2005/spring/survey_IntroSurveyMethods.pdf [Accessed 26/2/2013].

FERBER, J. 1999. *Multi-Agent Systems, An Introduction to Distributed Artificial Intelligence*, Addison-Wesley, ISBN 0-201-36048-9

FINE, G. A. & ELSBACH, K. D. 2000. Ethnography and Experiment in Social Psychological Theory Building. *Journal of Experimental Social Psychology,* Vol 36**,** pp 51-76.

FRANK, S. A. 2009. Evolutionary foundations of cooperation and group cohesion. *In:* LEVIN, S. A. (ed.) *Games, Groups, and the Global Good.* Berlin: Springer-Verlag, ISBN 978-3-540-85435-7

FYFE, C. 1996. Artificial Neural Networks. Dept. of Computing and Information Systems, University of Paisley, Scotland UK, Available http://cdn.preterhuman.net/texts/other/Neural%20Networks%20-%20Artificial%20Neural%20Networks%20-%20Collin%20Fyfe.pdf [Accessed 16/4/2012]

GHAVAMZADEH, M. & MAHADEVAN, S. 2004. Learning to Communicate and Act Using hierarchical Reinforcement Learning. *Third International Joint Conference on Autonomous Agents and Multi-agent Systems.* New York: IEEE Computer Society, Vol 3, pp 1114-1121

GILBERT, D., APARICIO & AL., E. 1995. The Role of Intelligent Agents in the Information Infrastructure. USA: IBM.

GILBERT, N. 1996. Simulation as a Research Strategy. *In:* TROITZSCH, K. G., MUELLER, U., GILBERT, G. N. & DORAN, J. E. (eds.) *Social science microsimulation.* Berlin: Springer, pp 448-454

GILBERT, N. 2007. Agent Based Models. *Quantitative Applications in the Social Sciences.* Sage Publications, pp 1-20, ISBN 9781412949644

GILBERT, N. & TROITZSCH, K. G. 2005. *Simulation for the Social Scientist*, Open University Press, ISBN 0335216005

GILLES, R. M. & ASHMAN, A. F. 2003. *Cooperative Learning: The social and intellectual outcomes of learning in groups,* London, Farmer Press, ISBN 978-0415303415

GOKHALE, A. A. 1995. Collaborative Learning Enhances Critical Thinking. *Journal of Technology Education,* Vol 7 No 1, ISBN 1045-1064

GOLDSPINK, C. 2002. Methodological Implications of Complex Systems Approaches to Sociality: Simulation as a foundation for Knowledge. *Journal of Artificial Societies and Social Simulation,* Vol 5 No 1.

GREEN, D. G. 1993. Emergent behaviour in biological systems. *In:* GREEN, D. G. &

BOSSOMAIER, T. J. (eds.) *Complex Systems - From Biology to Computation.* IOS Press, Amsterdam.

GROVES, R. M., FLOYD J. FOWLER, J., COUPER, M. P., LEPKOWSKI, J. M., SINGER, E. & TOURANGEAU, R. 2004. *Survey Methodology*, John Wiley & Sons.

GUGEL, S. A., PRATT, D. R. & SMITH, R. A. 2001a. The Implementation of Multiple Behavioural Approaches in the CFG Test-bed. 10th Conference on Computer Generated Forces & Behavioral Representation, 2001 Norfolk, VA.

GUGEL, S. A., PRATT, D. R. & SMITH, R. A. 2001b Using Q-Learning to Control Individual Combatants. 10th Conference on Computer Generated Forces & Behavioral Representation, 2001 Norfolk, VA.

HAMILTON, W. D. 1963. The evolution of altruistic behavior. *The American Naturalist,* Vol 97 No 896**,** pp 354-356.

HANSEN, E. A., BERNSTEIN, D. S. & ZILBERSTEIN, S. 2004. Dynamic Programming for Partially Observable Stochastic Games. Proceedings of the Nineteenth International Conference on Artificial Intelligence, July 25-29 2004 San Jose, CA. American Association for Artificial Intelligence, pp 709—715.

HARDY, C. 2005. *A STUDY OF MIDWEST STUDENTS' TECHNOLOGY SKILLS.* PhD, University of Nebraska. Available http://dwb4.unl.edu/Diss/Hardy/intro.pdf [Accessed 12/7/2012]

HENNINGER, A., GONZALEZ, A., GEORGIPOULOS, M. & DEMARA, R. 2001. The Limitations of Static Performance Metrics for Dynamic Tasks Learned Through Observation. 10th Conference on Computer Generated Forces & Behavioral Representation, 2001 Norfolk, VA.

HIRASHIMA, Y. 2008. A Q-Learning System for Container Transfer Scheduling Based on Shipping Order at Container Terminals. *International Journal of Innovative Computing, Information and Control,* Vol 4 No 3**,** pp 547-558.

HU, J. & WELLMAN, M. P. 1998. Multiagent Reinforcement Learning: theoretical framework and an algorithm. Proceedings of the Fifteenth International Conference on Machine Learning, San Francisco, CA. Morgan Kaufmann Publishers Inc, pp 242 – 250.

HUANG, B., CAO, G. & GUO, M. 2005. Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance. Proceedings of

Internal Conference on Machine Learning and Cybernetics, 2005 Guangzhou China, pp 85-89

HUNT, C. W. & SAIAS, I. 1998. Complexity-Based Modeling and Simulation: Modeling Innovation at the Edge of Chaos, Command and Control Research and Technology Symposium, 1998 Monterey, California.

IHRIG, M. A 2012, New Research Architecture For The Simulation Era. *In:* TROITZSCH, K. G., MOHRING, M. & LOTZMANN, U., eds. 26th European Conference on Modelling and Simulation, Koblenz Germany, ISBN 978-0-9564944-4-3

ILACHINSKI, A. 1996. Land Warfare and Complexity, Part II: An assessment of the Applicability of Non-linear Dynamics and Complex Systems Theory to the Study of Land Warfare. Alenandria, Virginia, USA: Center for Naval Analyses.

ILACHINSKI, A. 1997. Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Warfare (U). Alenandria, Virginia, USA: Center for Naval Analyses.

ILACHINSKI, A. 1999. Towards a Science of Experimental Complexity: An Artificial-Life Approach to Modeling Warfare. In: DING, M., DITTO, W., PECORA, L. M. & SPANO, M. L. (eds.) 5th Experimental Chaos Conference. Orlando, Florida.

JAIN, A. K., MAO, J. & MOHIUDDIN, K. 1996. Artificial Neural Networks: A Tutorial. *IEEE Computer,* Vol 29 No 3**,** pp 31-44.

JANG, J.S.R., SUN, C.T. & MIZUTANI, E. 1997. *Neuro-Fuzzy and Soft Computing*, Prentice Hall, ISBN 0-13-261066-3

JENNINGS, N. R. 2001. An agent-based approach for building complex software systems, *Communications of the ACM* Vol 44 No. 4 pp 35-41

JOHANSSON, R. 2003. Case Study Methodology : Keynote. *Methodologies in Housing Research.* Stockholm, Available http://www.infra.kth.se/BBA/IAPS%20PDF/paper%20Rolf%20Johansson%20ver%202.pdf [Accessed 2/4/2008]

JOHNSON, D. W., MARUYAMA, G., JONNSON, R. T., NELSON, D. & SKON, L. 1981. Effects of cooperative, competitive, and individualistic goal structures on achievement: A meta analysis.: Psychological Bulletin, Vol 89, pp 47-62.

JOHNSON, D. W. & JOHNSON, R.T. 1985. The Internal Dynamics of Cooperative

Learning Groups. *In:* SALVIN, R., SHARAN, S., KAGAN, S., LAZAROWITZ, R. H., WEBB, C. & SCHMUCK, R. (eds.) *Learning To Cooperate, Cooperating To Learn.* New York: Plenum Press.

KANNENGIESSER, U. & GERO, J.S.. 2005 "Understanding Situated Agents" Proceedings of the CAADRIA'05 Vol 2, TVB New Delhi, pp. 277-287.

KELLY, I. D. & KEATING, D. A. 1998. Increased Learning Rates Through the Sharing of Experiences of Multiple Autonomous Mobile Robot Agents. Proceedings of the 7th IEEE International Conference on Fuzzy Systems, May 1998 Anchorage, Alaska, pp 129-134.

KHOUSSAINOV, R. 2004. Towards Well-defined Multi-agent Reinforcement Learning. *In:* BUSSLER, C. & FENSEL, D., eds. Proceedings of the 11th International Conference on Artificial Intelligence: Methodology, Systems, and Applications, 2004 Varna, Bulgaria. Springer, pp 399-408.

KOTHARI, C. R. 2004. *Research Methodology: Methods and Techniques,* Delhi, India, New Age International Publishers. Available http://www.limat.org/data/research/Research%20Methodology.pdf [Accessed 19/03/2012]

KROSE, B. & VAN DER SMAGT, P. 1996. *An Introduction to Neural Networks,* The Netherlands, Faculty of Mathematics and COmputer Science, University of Amsterdam Available http://lia.univ-avignon.fr/fileadmin/documents/Users/Intranet/chercheurs/torres/livres/book-neuro-intro.pdf [Accessed] 23 5/2013]

LAPIN, B. D., KOLBE, R. L., LANGWORTHY, J., TRAN, V. & KANG, R. Semi-Automated Forces that Learn from Experience. 10th Conference on Computer Generated Forces & Behavioral Representation, 2001 Norfolk, VA.

LEE, E. A. 2006. The Problem with Threads. *IEEE Computer,* Vol 39 (5), 33-42.

LEEDY, P. D. 1997. *Practical Research*, Prentice Hall, ISBN 0-13-241407-4

LEEDY, P. D. & ORMROD, J. E. 2001. *Practical Research: Planning and Design*. ISBN 9780139603600

LITTMAN, M. L. Markov Games as a Framework for Multi-agent Reinforcement Learning. Proceedings of the 11th International Conference on Machine Learning, 1994 1994 New Brunswick, NJ. Morgan Kaufmann, pp 157-163.

LUCAS, C. 2000. Perturbation and Transients - The Edge of Chaos. CALRESCO, Availavle http://www.calresco.org/perturb.htm [Accessed 2/4/2000]

LUZZI, J. 2005a. The Case Study. Available: http://www.kean.edu/~jluzzi/classes/Kn-case.doc [Accessed 28/2/2013].

LUZZI, J. 2005b. Experimental Research. Available: www.kean.edu/~jluzzi/classes/experim.doc [Accessed 28/2/2013].

MACAL, C. M. & NORTH, M. J. 2005. Tutorial on agent-based modeling and simulation. Proceedings of the 37th Winter Simulation conference, 2005, Orlando FL, ISBN 0-7803-9519-0, pp 2-15

MARCENAC, P. 1996. Emergence of Behaviours in Natural Phenomena Agent-Simulation. *Complexity International,* Vol 3, Available http://www.csu.edu.au/ci/vol03/cs964/cs964.html [Accessed 6/9/2012]

MATARIC, M. J. 1998. Using communication to reduce locality in distributed multiagent learning. *Journal of Experimental & Theoretical Artificial Intelligence,* Vol 10 No 3**,** pp 357-369.

MAY, M. A. & DOOB, L. W. 1937. Cooperation and Competition. New York: Social Science Research Bulletin, No 25

MAZHER, A. K. 2001. Towards a Unified Approach to Modeling Computer Simulation of Social Systems, Part I. *2nd Workshop on Norms and Institutions in multi-Agent Systems at The 5th International Conference on Autonomous Agents.* Montreal, Canada: ACM, New York, USA.

MCCULLOCH, W. S. & PITTS, W. 1943. A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics,* Vol 5**,** pp 115-133.

MCHANEY, R. 1991. *Computer Simulation: a practical perspective*, Academic Press, ISBN 0-12-484140-6

MCHANEY, R. 2009. Understanding Computer Simulation. Ventus Publishing ApS, ISBN 978-87-7681-505-9

MCKAY, J. & MARSHALL, P. 2001. The Dual Imperatives of Action Research. *Information Technology & People,* Vol 14 No 1**,** pp 46-59.

MICHEL, F., FERBER, J. & DROGOUL, A. 2009. Multi-Agent Systems and

Simulation: a Survey From the Agents Community's Perspective. *In:* UHRMACHER, A. M. & WEYNS, D. (eds.) *Multi-Agent Systems: Simulation and Applictions.* CRC Press.

MILIJKOVIC, Z., MITIC, M., LAZAREVIC, M. & BABIC, B. 2013. Neural network Reinforcement Learning for Visual control of Robot manipulators. *Expert Systems with Applications,* Vol 40 No. 5 Aprin 2013 pp 1721-1736

MITCHELL, T. M. 1997a. *Machine Learning*, McGraw-Hill International Editions, ISBN 0-07-115467-1

MITCHELL, T. M. 1997b. Reinforcement Learning. *In:* Liu, C. I. (Series ed.) *Machine Learning.* Computer Science Series, Mc Graw-Hill International Editions. ISBN 0 07 115467 1

NAKANO, N. 2007. A Pilot Study on Modeling and Analysis of Capricious Agents in Agent-based Simulations. *International Journal of Innovative Computing, Information and Control,* Vol No 14**,** pp 91-100.

NICKLES, M., ROVATSOS, M., BRAUER, W. & WEIB, G. 2004. Towards a unified model of sociality in multiagent systems. *International Journal of Computer and Information Science (IJCIS),* Vol 5 No 2**,** pp 73-88.

NUNES, L. & OLIVEIRA, E. 2003. On learning by Exchanging Advice. *Interdisciplinary Journal of Artificial Intelligence and the Simulation of Behaviour,* Vol 1 No 3 **,** pp 241.

NWANA, H. 1996. Software Agents: An Overview. *Knowledge Engineering Review,* Vol 11 No 3.

ODELL, J. 2010. Agent Technology: An Overview, Computer Sciences Corporation Available: http://www.jamesodell.com/Agent_Technology-An_Overview.pdf [Accessed 27/11/2012].

O'DELL, R. & WATTENHOFER, R. 2005. Information Dissemination in Highly Dynamic Graphs. *2005 Joint Workshop on Foundations for Mobile Computing.* Cologne, Germany: ACM, ISBN 1-59593-092-2

PAPAHRISTOU, N. & REFANIDIS, I. 2011. Training Neural Networks to Play Backgammon Variants Using Reinforcement Learning. L*ecture Notes in Computer Science 6624,* Part 1 pp 113-122, Springer-Verlag

PARSONS, S., MCBURNEY, P. & SKLAR, E. 2010. Reasoning about Trust using

Argumentation: A position paper. 7th Workshop on Argumentation in Multiagent Systems. Toronto.

PERROS, H. 2009. Computer Simulation Techniques: The definitive introduction! Raleigh, North Carolina: Computer Science Department, North Carolina State University, Available http://www.csc.ncsu.edu/faculty/perros//simulation.pdf [Accessed 21/12/2011]

PETTY, M. D. 2001. Do We Really Want Computer Generated Forces That Learn? 10th Conference on Computer Generated Forces & Behavioral Representation, 2001 Norfolk, VA.

PREUX, P., DELEPOULLE, S. & DARCHEVILLE, J. 2004. A Generic Architecture for Adaptive Agents Based on Reinforcement Learning. *Information Sciences,* Vol 161 No 102 **,** pp 37-55.

PRICE, B. & BOUTILIER, C. 1999. Implicit Imitation in Multiagent Reinforcement Learning. Proceedings of the Sixteenth International Conference on Machine Learning, 1999 Bled, Slovenia. Morgan Kaufmann Publishers Inc, pp 325 – 334. ISBN 1-55860-612-2

RAMANATH, A. M. & GILBERT, N. 2003. Towards a Methodology for Agent Based Social Simulation Research. In proceedings *AGENTLINK.* Barcelona, Spain.

REITMAN, J. 1988. A concise history of the ups and downs of simulation. *In:* ABRAMS, M., HAIGH, P. & COMFORT, J. (eds.) *1988 Winter Simulation Conference* San Diego: ACM.

RESTLE, F. & DAVIS, J. 1962. Success and speed of problem solving by individuals and groups. *Psychological Review,* Vol 69 Issue 6 Nov 1962 **,** pp 520-536.

REYNOLDS, C. W. (1987) Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34.

REZAEI, G. 2011. *Enhanced social learning via trust and reputation mechanisms in multi-agent systems.* PhD, University of Melbourne, Australia.

ROJAS, R. 1996. *Neural Networks: A Systematic Introduction,* Berlin, Springer-Verlag, Available http://page.mi.fu-berlin.de/rojas/neural/neuron.pdf [Accessed 16/9/2012]

RUAS, T. L., MARIETTO, M., BATISTA, A., FRANCA, R., HIEDEKER, A.,

NORONHA, E. & DA SILVA, F. 2011. Modeling Artificial Life Through Multi-Agent Based Simulation. *In:* ALKHATEEB, F., AL MAGHAYREH, E. & DOUSH, I. A. (eds.) *Multi-Agent Systems - Modeling, Control, Programming, Simulations and Applications*. Rijeka, Croatia: InTech, pp41-66, ISBN 978-953-307-174-9

RUBENSTEIN, D. & KEALEY, J. 2010. Cooperation, Conflict, and the Evolution of Complex Animal Societies. *Nature Education Knowledge 1(8):47* [Online]. Available: http://www.nature.com/scitable/knowledge/library/cooperation-conflict-and-the-evolution-of-complex-13236526 [Accessed 2/4/2011].

SAOUD, N. B. & MARK, G. 2007. Complexity Theory and Collaboration: An Agent-based Simulator for a Space Mission Design Team. *Computational & Mathematical Organization Theory,* Vol 13 No 2**,** pp 113-146.

SEN, S., SEKARAN, M. & HALE, J. 1994. Learning to Coordinate Without Sharing Information. Proceedings of the twelfth national conference on Artificial intelligence, 1994 Menlo Park, CA. American Association for Artificial Intelligence, Vol 1, pp 426 – 431.

SHAVELSON, R. J. & TOWNE, L. 2002. Scientific Research in Education. Washington DC: Center for Education, National Research Council, ISBN 0-309-08291-9

SHAW, M. E. 1932. A Comparison of Individuals and Small Groups in the Rational Solution of Complex Problems. *The American Journal of Psychology,* Vol 44 No 3**,** pp 491-504.

SHOHAM, Y., POWERS, R. & GRENAGER, T. 2003. Multi-agent Reinforcement Learning: A Critical Survey. Computer Science Department, Stanford University, Report ISBN 2/3/2005 Available http://jmvidal.cse.sc.edu/library/shoham03a.pdf [Accessed 2/7/2013]

SIAN, S. Extending Learning to Multiple Agents: Issues and a Model for Multi-Agent Machine Learning. *In:* KODRATOFF, Y., ed. Machine Learning - EWSL-91, 1991 Porto, Portugal. Springer-Verlag, 440-456.

SIMON, M., DR. 2011. Dissertations and Scholarly Research. Available: http://dissertationrecipes.com/wp-content/uploads/2011/04/AssumptionslimitationsdelimitationsX.pdf [Accessed 28/02/2013].

SKRZYPCZYNSKI, P. 2004. Managing the Communication in a Complex System of Robots and Sensors. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.268 [Accessed 3/10/2012]

SMITH, C. U. M. 1995. The Complexity of Brains: A Biologist's View. *Complexity International,* Vol 2.

STERELNY, K. 2008. The Human Co-operation Syndrome. Available: http://www.institutnicod.org/Backgroundto_lecture4.pdf [Accessed 24/2/2011].

STONE, P. & VELOSO, M. 1997. Multi-agent Systems: A Survey from a Machine Learning Perspective. Pittsburgh, PA: Carnegie Mellon University, School of Computer Science, Report No. CMU-CS-97-193

STANGROOM, J 2014. Social Science Statistics. Available: http://www.socscistatistics.com/tests/Default.aspx [Accessed 20/6/2014].

SUN, R. 2001. Cognitive Science Meets Multi-Agent Systems: A Prolegomemon. *Philosophical Psychology,* Vol 14 No 1**,** pp 5-28.

SUNTZU 2005. The Art of War, El Paso, Texas, El Paso Norte Press, Translated by Giles, L., ISBN 1-934255-12-2.

SUTTON, R.S. 1988. Neural network Reinforcement Learning to Predict by the methods of Temporal Difference. *Machine Learning,* Vol 3 1988 pp 9-44

SUTTON, R. S. & BARTO, A. G. 1998. *Reinforcement Learning: An Introduction,* Cambridge, MA, MIT Press, ISBN 0-262-19398-1

SYCARA, P. 1998. Multiagent Systems. *AI Magazine,* Vol 19 No 2 **,** pp 79-92.

TAN, M. 1993. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. *In:* HUHNS, M. N. & SINGH, M. P. (eds.) *Readings in agents.* San Francisco, CA: Morgan Kaufmann Publishers Inc.

TERÁN, O., EDMONDS, B. & WALLIS, S. 2000. Determining the Envelope of Emergent Agent Behaviour via Architectural Transformation. 4th International Conference on MultiAgent Systems (ICMAS2000), Boston, MA.

TESAURO, G. 1995. Itemporal Difference Learning and TD-Gammon. *Communications of the ACM,* Vol 38 No 3.

VAN DER HOOG, S. 2004. On Multi-Agent Based Simulation. Available: http://www.wiwi.uni-bielefeld.de/fileadmin/vpl1/mabs-survey.pdf [Accessed 3/2/2011].

VAUGHAN, J. & CONNELL, R. 2000 "Combining Intelligent Agents & Artificial Intelligence for the Land Force," Proceedings of the SimTecT 2000 Conference, Sydney, AU, 28 February – 2 March, pp. 77-82.

VERSINO, C. & GAMBARDELLA, L. M. 1997. Learning real Team Solutions. *In:* WEISS, G. (ed.) *Distributed Artificial Intelligence meets machine Learning, Learning in Multi-Agent Environments.* Springer. 1221, Lecture Notes in Computer Science pp 40-61

WALLIMAN, N. 2005. *Your Research Project: A step-by-step guide for the first-time researcher*, Sage Study Skills Series. SAGE, ISBN 9781446200179

WATKINS, C. J. C. H. 1989. *Learning from Delayed Rewards.* Cambridge University, PhD Thesis, Available http://www.cs.rhul.ac.uk/home/chrisw/new_thesis.pdf [Accessed 4/3/2013]

WOLPERT, D. H. & TUMER, K. 2000. An Introduction to Collective Intelligence. NASA, Report NASA tech rep NASA-ARC-IC-99-63 Available http://www.cse.msu.edu/rlr/pub/Wolpert1.html [Accessed 11/8/2011]

WOOLDRIDGE, M. & JENNINGS, N. 1995. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review,* Vol 10 No 2.

ZAMBONELLI, F., JENNINGS, N. & WOOLDRIDGE, M.. 2005. Multi-Agent Systems as Computational Organizations: The Gaia Methodology. *In:* HENDERSON-SELLERS, B. & GIORGINI, P. (ed.) *Agent-Oriented Methodologies.* Idea Group pp 136-171

# APPENDIX A

# SIMULATION CODE

*Controlling complexity is the essence of computer programming*

– *Brian Kernighan*

The following pages in this Appendix contain the code for the Java classes used in the implementation of the Multi-agent simulation tool developed and discussed in Chapter 4 Experimental Design.

## A.1 Ares Application Main Class

```
/*
 * AresApp.java
 */

package ares;

import org.jdesktop.application.Application;
import org.jdesktop.application.SingleFrameApplication;

/**
 * The main class of the application.
 */
public class AresApp extends SingleFrameApplication {

    /**
     * At startup create and show the main frame of the application.
     */
    @Override protected void startup() {
        show(new AresView(this));
    }

    /**
     * This method is to initialize the specified window by injecting resources.
     * Windows shown in our application come fully initialized from the GUI
     * builder, so this additional configuration is not needed.
     */
    @Override protected void configureWindow(java.awt.Window root) {
    }

    /**
     * A convenient static getter for the application instance.
     * @return the instance of AresApp
     */
    public static AresApp getApplication() {
        return Application.getInstance(AresApp.class);
    }

    /**
     * Main method launching the application.
     */
    public static void main(String[] args) {
        launch(AresApp.class, args);
    }
}
```

## A.2 Ares Application Main GUI Form Class

```
/*
 * AresView.java
 */
package ares;

import org.jdesktop.application.Action;
import org.jdesktop.application.ResourceMap;
import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import org.jdesktop.application.TaskMonitor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Timer;
import javax.swing.Icon;
import javax.swing.JDialog;
import javax.swing.JFrame;
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

/**
 * The application's main frame.
 */
public class AresView extends FrameView {

    JFrame gui;
    boolean OK = false;
    private String paramFile;

    public AresView(SingleFrameApplication app) {
        super(app);

        initComponents();

        initialize();  // my own initialization

        // status bar initialization - message timeout, idle icon and busy animation, etc
        ResourceMap resourceMap = getResourceMap();
        int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
        messageTimer = new Timer(messageTimeout, new ActionListener() {

            public void actionPerformed(ActionEvent e) {
                statusMessageLabel.setText("");
            }
        });
        messageTimer.setRepeats(false);
        int busyAnimationRate = resourceMap.getInteger("StatusBar.busyAnimationRate");
        for (int i = 0; i < busyIcons.length; i++) {
            busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]");
```

```
    }
    busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {

        public void actionPerformed(ActionEvent e) {
            busyIconIndex = (busyIconIndex + 1) % busyIcons.length;
            statusAnimationLabel.setIcon(busyIcons[busyIconIndex]);
        }
    });
    idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
    statusAnimationLabel.setIcon(idleIcon);
    progressBar.setVisible(false);

    // connecting action tasks to status bar via TaskMonitor
    TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
    taskMonitor.addPropertyChangeListener(new java.beans.PropertyChangeListener() {

        public void propertyChange(java.beans.PropertyChangeEvent evt) {
            String propertyName = evt.getPropertyName();
            if ("started".equals(propertyName)) {
                if (!busyIconTimer.isRunning()) {
                    statusAnimationLabel.setIcon(busyIcons[0]);
                    busyIconIndex = 0;
                    busyIconTimer.start();
                }
                progressBar.setVisible(true);
                progressBar.setIndeterminate(true);
            } else if ("done".equals(propertyName)) {
                busyIconTimer.stop();
                statusAnimationLabel.setIcon(idleIcon);
                progressBar.setVisible(false);
                progressBar.setValue(0);
            } else if ("message".equals(propertyName)) {
                String text = (String) (evt.getNewValue());
                statusMessageLabel.setText((text == null) ? "" : text);
                messageTimer.restart();
            } else if ("progress".equals(propertyName)) {
                int value = (Integer) (evt.getNewValue());
                progressBar.setVisible(true);
                progressBar.setIndeterminate(false);
                progressBar.setValue(value);
            }
        }
    });
}
//*****************  My Stuff *********************************************

private void initialize() {
    gui = this.getFrame();
    gui.setTitle("ARES Simulation User Interface");
    paramFile = null;

    //disable some menus and menu items
    enableSimMenu(false);
    enableParamMenu(false);
    enableCloseFileMi(false);
```

```
        enableSaveasFileMi(false);
        enableSaveFileMi(false);
    }

    private void enableSimMenu(boolean b) {
        if (b) {
            simulationM.setEnabled(true);
        } else {
            simulationM.setEnabled(false);
        }
    }

    private void enableParamMenu(boolean b) {
        if (b) {
            parametersM.setEnabled(true);
        } else {
            parametersM.setEnabled(false);
        }
    }

    private void enableCloseFileMi(boolean b) {
        if (b) {
            closeFileMi.setEnabled(true);
        } else {
            closeFileMi.setEnabled(false);
        }
    }

    private void enableSaveFileMi(boolean b) {
        if (b) {
            saveFileMi.setEnabled(true);
        } else {
            saveFileMi.setEnabled(false);
        }
    }

    private void enableSaveasFileMi(boolean b) {
        if (b) {
            saveasFileMi.setEnabled(true);
        } else {
            saveasFileMi.setEnabled(false);
        }
    }

//**************************************************************************
*
    @Action
    public void showAboutBox() {
        if (aboutBox == null) {
            JFrame mainFrame = AresApp.getApplication().getMainFrame();
            aboutBox = new AresAboutBox(mainFrame);
            aboutBox.setLocationRelativeTo(mainFrame);
        }
        AresApp.getApplication().show(aboutBox);
    }
```

```
    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        mainPanel = new javax.swing.JPanel();
        jButton1 = new javax.swing.JButton();
        menuBar = new javax.swing.JMenuBar();
        javax.swing.JMenu fileMenu = new javax.swing.JMenu();
        loadFileMi = new javax.swing.JMenuItem();
        newFileMi = new javax.swing.JMenuItem();
        saveasFileMi = new javax.swing.JMenuItem();
        saveFileMi = new javax.swing.JMenuItem();
        closeFileMi = new javax.swing.JMenuItem();
        jSeparator1 = new javax.swing.JPopupMenu.Separator();
        javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();
        parametersM = new javax.swing.JMenu();
        getSimParamMi = new javax.swing.JMenuItem();
        getWorldParamMi = new javax.swing.JMenuItem();
        getAgentParamMi = new javax.swing.JMenuItem();
        jSeparator2 = new javax.swing.JPopupMenu.Separator();
        checkParamGrpsMi = new javax.swing.JMenuItem();
        simulationM = new javax.swing.JMenu();
        startSimMi = new javax.swing.JMenuItem();
        jMenu1 = new javax.swing.JMenu();
        javax.swing.JMenu helpMenu = new javax.swing.JMenu();
        javax.swing.JMenuItem aboutMenuItem = new javax.swing.JMenuItem();
        statusPanel = new javax.swing.JPanel();
        javax.swing.JSeparator statusPanelSeparator = new javax.swing.JSeparator();
        statusMessageLabel = new javax.swing.JLabel();
        statusAnimationLabel = new javax.swing.JLabel();
        progressBar = new javax.swing.JProgressBar();

        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(ares.AresApp.class).getContext().getResource
Map(AresView.class);
        mainPanel.setBackground(resourceMap.getColor("mainPanel.background")); // NOI18N
        mainPanel.setName("mainPanel"); // NOI18N

        jButton1.setBackground(resourceMap.getColor("jButton1.background")); // NOI18N
        jButton1.setForeground(resourceMap.getColor("jButton1.foreground")); // NOI18N
        jButton1.setIcon(resourceMap.getIcon("jButton1.icon")); // NOI18N
        jButton1.setText(resourceMap.getString("jButton1.text")); // NOI18N
        jButton1.setBorder(null);
        jButton1.setBorderPainted(false);
        jButton1.setContentAreaFilled(false);
        jButton1.setDebugGraphicsOptions(javax.swing.DebugGraphics.NONE_OPTION);
        jButton1.setFocusPainted(false);
        jButton1.setHideActionText(true);
```

```
        jButton1.setName("jButton1"); // NOI18N

        javax.swing.GroupLayout mainPanelLayout = new javax.swing.GroupLayout(mainPanel);
        mainPanel.setLayout(mainPanelLayout);
        mainPanelLayout.setHorizontalGroup(
            mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addComponent(jButton1)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        mainPanelLayout.setVerticalGroup(
            mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addComponent(jButton1)
                .addContainerGap(14, Short.MAX_VALUE))
        );

        menuBar.setBorder(new
javax.swing.border.SoftBevelBorder(javax.swing.border.BevelBorder.RAISED));
        menuBar.setName("menuBar"); // NOI18N

        fileMenu.setText(resourceMap.getString("fileMenu.text")); // NOI18N
        fileMenu.setName("fileMenu"); // NOI18N

        loadFileMi.setText(resourceMap.getString("loadFileMi.text")); // NOI18N
        loadFileMi.setName("loadFileMi"); // NOI18N
        loadFileMi.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                loadFileMiActionPerformed(evt);
            }
        });
        fileMenu.add(loadFileMi);

        newFileMi.setText(resourceMap.getString("newFileMi.text")); // NOI18N
        newFileMi.setName("newFileMi"); // NOI18N
        newFileMi.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                newFileMiActionPerformed(evt);
            }
        });
        fileMenu.add(newFileMi);

        saveasFileMi.setText(resourceMap.getString("saveasFileMi.text")); // NOI18N
        saveasFileMi.setName("saveasFileMi"); // NOI18N
        fileMenu.add(saveasFileMi);

        saveFileMi.setText(resourceMap.getString("saveFileMi.text")); // NOI18N
        saveFileMi.setName("saveFileMi"); // NOI18N
        fileMenu.add(saveFileMi);

        closeFileMi.setText(resourceMap.getString("closeFileMi.text")); // NOI18N
        closeFileMi.setName("closeFileMi"); // NOI18N
        closeFileMi.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                closeFileMiActionPerformed(evt);
```

```
        }
    });
    fileMenu.add(closeFileMi);

    jSeparator1.setName("jSeparator1"); // NOI18N
    fileMenu.add(jSeparator1);

    javax.swing.ActionMap actionMap =
org.jdesktop.application.Application.getInstance(ares.AresApp.class).getContext().getActionMap
(AresView.class, this);
    exitMenuItem.setAction(actionMap.get("quit")); // NOI18N
    exitMenuItem.setName("exitMenuItem"); // NOI18N
    fileMenu.add(exitMenuItem);

    menuBar.add(fileMenu);

    parametersM.setText(resourceMap.getString("parametersM.text")); // NOI18N
    parametersM.setName("parametersM"); // NOI18N

    getSimParamMi.setText(resourceMap.getString("getSimParamMi.text")); // NOI18N
    getSimParamMi.setName("getSimParamMi"); // NOI18N
    getSimParamMi.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            getSimParamMiActionPerformed(evt);
        }
    });
    parametersM.add(getSimParamMi);

    getWorldParamMi.setText(resourceMap.getString("getWorldParamMi.text")); // NOI18N
    getWorldParamMi.setName("getWorldParamMi"); // NOI18N
    getWorldParamMi.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            getWorldParamMiActionPerformed(evt);
        }
    });
    parametersM.add(getWorldParamMi);

    getAgentParamMi.setText(resourceMap.getString("getAgentParamMi.text")); // NOI18N
    getAgentParamMi.setName("getAgentParamMi"); // NOI18N
    getAgentParamMi.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            getAgentParamMiActionPerformed(evt);
        }
    });
    parametersM.add(getAgentParamMi);

    jSeparator2.setName("jSeparator2"); // NOI18N
    parametersM.add(jSeparator2);

    checkParamGrpsMi.setText(resourceMap.getString("checkParamGrpsMi.text")); // NOI18N
    checkParamGrpsMi.setName("checkParamGrpsMi"); // NOI18N
    checkParamGrpsMi.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            checkParamGrpsMiActionPerformed(evt);
        }
```

```
        });
        parametersM.add(checkParamGrpsMi);

        menuBar.add(parametersM);

        simulationM.setText(resourceMap.getString("simulationM.text")); // NOI18N
        simulationM.setName("simulationM"); // NOI18N

        startSimMi.setText(resourceMap.getString("startSimMi.text")); // NOI18N
        startSimMi.setName("startSimMi"); // NOI18N
        startSimMi.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                startSimMiActionPerformed(evt);
            }
        });
        simulationM.add(startSimMi);

        menuBar.add(simulationM);

        jMenu1.setForeground(resourceMap.getColor("jMenu1.foreground")); // NOI18N
        jMenu1.setText(resourceMap.getString("jMenu1.text")); // NOI18N
        jMenu1.setContentAreaFilled(false);
        jMenu1.setEnabled(false);
        jMenu1.setName("jMenu1"); // NOI18N
        menuBar.add(jMenu1);

        helpMenu.setText(resourceMap.getString("helpMenu.text")); // NOI18N
        helpMenu.setName("helpMenu"); // NOI18N

        aboutMenuItem.setAction(actionMap.get("showAboutBox")); // NOI18N
        aboutMenuItem.setName("aboutMenuItem"); // NOI18N
        helpMenu.add(aboutMenuItem);

        menuBar.add(helpMenu);

        statusPanel.setName("statusPanel"); // NOI18N

        statusPanelSeparator.setName("statusPanelSeparator"); // NOI18N

        statusMessageLabel.setName("statusMessageLabel"); // NOI18N

        statusAnimationLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        statusAnimationLabel.setName("statusAnimationLabel"); // NOI18N

        progressBar.setName("progressBar"); // NOI18N

        javax.swing.GroupLayout statusPanelLayout = new
javax.swing.GroupLayout(statusPanel);
        statusPanel.setLayout(statusPanelLayout);
        statusPanelLayout.setHorizontalGroup(
            statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(statusPanelSeparator, javax.swing.GroupLayout.DEFAULT_SIZE, 510,
Short.MAX_VALUE)
            .addGroup(statusPanelLayout.createSequentialGroup()
                .addContainerGap()
```

```
                .addComponent(statusMessageLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 340,
Short.MAX_VALUE)
                .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(statusAnimationLabel)
                .addContainerGap())
        );
        statusPanelLayout.setVerticalGroup(
            statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(statusPanelLayout.createSequentialGroup()
                .addComponent(statusPanelSeparator, javax.swing.GroupLayout.PREFERRED_SIZE,
2, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                    .addComponent(statusMessageLabel)
                    .addComponent(statusAnimationLabel)
                    .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(3, 3, 3))
        );

        setComponent(mainPanel);
        setMenuBar(menuBar);
        setStatusBar(statusPanel);
        addPropertyChangeListener(new java.beans.PropertyChangeListener() {
            public void propertyChange(java.beans.PropertyChangeEvent evt) {
                formPropertyChange(evt);
            }
        });
    }// </editor-fold>//GEN-END:initComponents

    private void formPropertyChange(java.beans.PropertyChangeEvent evt) {//GEN-
FIRST:event_formPropertyChange
        // TODO add your handling code here:
    }//GEN-LAST:event_formPropertyChange

    private void closeFileMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_closeFileMiActionPerformed
        // TODO add your handling code here:
        paramFile = null;
        enableSimMenu(false);
        enableParamMenu(false);
        enableCloseFileMi(false);
    }//GEN-LAST:event_closeFileMiActionPerformed

    private void newFileMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_newFileMiActionPerformed
        // TODO add your handling code here:
        String newFileName;
        RandomAccessFile f;
```

```
        int yesNoC;

        newFileName = JOptionPane.showInputDialog(gui, "Please enter the name for the new
parameter file");
        if (newFileName == null) {
            return;
        }
        newFileName = AresUtil.makeFileName(newFileName);
        try {
            f = new RandomAccessFile(newFileName, "r");
            f.close();
            JOptionPane.showMessageDialog(gui, "A parameter file with this name already exists,
Please specify a new name", "Error Mesage", JOptionPane.ERROR_MESSAGE);
            return;
        } catch (IOException e) {
            yesNoC = JOptionPane.showConfirmDialog(gui, "Create new parameter file " +
newFileName + "?");
            if (yesNoC != 0) {
                return;
            }
            AresUtil.newParamFile(newFileName);
            paramFile = newFileName;
            enableParamMenu(true);
            enableCloseFileMi(true);
        }
    }//GEN-LAST:event_newFileMiActionPerformed

    private void loadFileMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_loadFileMiActionPerformed
        // TODO add your handling code here:
        JFileChooser chooser = new JFileChooser();
        chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        chooser.setDialogTitle("Select Parameter FIle");
        chooser.setCurrentDirectory(new File(AresUtil.paramFileDir));
        int returnVal = chooser.showOpenDialog(gui);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            paramFile = chooser.getSelectedFile().getPath();
            enableParamMenu(true);
            enableCloseFileMi(true);
        }
    }//GEN-LAST:event_loadFileMiActionPerformed

    private void getSimParamMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_getSimParamMiActionPerformed
        simPGUI dbox = new simPGUI(gui, true, paramFile, "Simulation");
        dbox.setVisible(true);
    }//GEN-LAST:event_getSimParamMiActionPerformed

    private void getWorldParamMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_getWorldParamMiActionPerformed
        worldPGUI dbox = new worldPGUI(gui, true, paramFile, "World");
        dbox.setVisible(true);
    }//GEN-LAST:event_getWorldParamMiActionPerformed

    private void getAgentParamMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
```

```
FIRST:event_getAgentParamMiActionPerformed
        agentLocSetupGUI dbox = new agentLocSetupGUI(gui, true, paramFile, "Agents");
        dbox.setVisible(true);
    }//GEN-LAST:event_getAgentParamMiActionPerformed

    private void checkParamGrpsMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_checkParamGrpsMiActionPerformed
        checkPGUI dbox = new checkPGUI(gui, true, paramFile);
        dbox.setVisible(true);
        OK = dbox.allOK;
        dbox.dispose();
        if (OK) {
            enableSimMenu(true);
        }
    }//GEN-LAST:event_checkParamGrpsMiActionPerformed

    private void startSimMiActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_startSimMiActionPerformed
        SimEngine s = new SimEngine(paramFile, gui);
        gui.setVisible(false);
           new Thread(s).start();
    }//GEN-LAST:event_startSimMiActionPerformed

    // Variables declaration - do not modify//GEN-BEGIN:variables
    javax.swing.JMenuItem checkParamGrpsMi;
    javax.swing.JMenuItem closeFileMi;
    javax.swing.JMenuItem getAgentParamMi;
    javax.swing.JMenuItem getSimParamMi;
    javax.swing.JMenuItem getWorldParamMi;
    javax.swing.JButton jButton1;
    javax.swing.JMenu jMenu1;
    javax.swing.JPopupMenu.Separator jSeparator1;
    javax.swing.JPopupMenu.Separator jSeparator2;
    javax.swing.JMenuItem loadFileMi;
    javax.swing.JPanel mainPanel;
    javax.swing.JMenuBar menuBar;
    javax.swing.JMenuItem newFileMi;
    javax.swing.JMenu parametersM;
    private javax.swing.JProgressBar progressBar;
    javax.swing.JMenuItem saveFileMi;
    javax.swing.JMenuItem saveasFileMi;
    javax.swing.JMenu simulationM;
    javax.swing.JMenuItem startSimMi;
    private javax.swing.JLabel statusAnimationLabel;
    private javax.swing.JLabel statusMessageLabel;
    javax.swing.JPanel statusPanel;
    // End of variables declaration//GEN-END:variables
    private final Timer messageTimer;
    private final Timer busyIconTimer;
    private final Icon idleIcon;
    private final Icon[] busyIcons = new Icon[15];
    private int busyIconIndex = 0;
    private JDialog aboutBox;
}
```

## A.3 Parameter Group Checking Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * checkPGUI.java
 *
 * Created on 27/11/2010, 7:09:01 PM
 */
package ares;

/**
 *
 * @author Paul
 */
import java.awt.Toolkit;
import java.io.IOException;
import java.io.RandomAccessFile;
import javax.swing.JOptionPane;


public class checkPGUI extends javax.swing.JDialog {

    boolean allOK = false;
    private String paramFile;
    private long hfp;     // pointer to parameter groups header record
    private long gfp;    // pointer to parameter groups data space
    private RandomAccessFile f;

    // parameter group variables
    private int width = 0, height = 0;

    /** Creates new form checkPGUI */
    public checkPGUI(java.awt.Frame parent, boolean modal, String file) {
        super(parent, modal);
        initComponents();

        this.setLocation(200,200);
        paramFile = file;
        checkParameters();
    }

          void checkParameters() {
                    String groupName, status;
                boolean groupStatus;

               long fp;
               int pGroups, i;

                try {
```

```
                f = new RandomAccessFile(paramFile,"rwd");
                    f.seek(100);
                    pGroups = f.readInt();

                      fp = 200;
                      allOK = true;
                      for (i = 1; i <= pGroups; i++) {
                        f.seek(fp);
                        groupName = f.readUTF();

                          f.seek(fp+40);
                          groupStatus = f.readBoolean();
                          if (groupStatus)
                              status = "completed";
                          else {
                              status = "not completed";
                              allOK = false;
                          }
                          outAreaTxt.append(groupName+"   "+status+"\n");
                          fp = fp + 50;
                      }

            }
            catch (IOException e) {
                        JOptionPane.showMessageDialog(null,"Trouble accessing the parameter
file","Error Mesage",JOptionPane.ERROR_MESSAGE);
                      exitDialog();
                    }
        }

        public void exitDialog() {
            try {
              f.close();
            }
            catch (IOException e) {
              JOptionPane.showMessageDialog(this,"An error occurred closing the parameter
file","Error Mesage",JOptionPane.ERROR_MESSAGE);
            }
            setVisible(false);
        }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        okBtn = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        outAreaTxt = new javax.swing.JTextArea();
```

```
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setName("Form"); // NOI18N
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosed(java.awt.event.WindowEvent evt) {
                formWindowClosed(evt);
            }
        });

        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(ares.AresApp.class).getContext().getResource
Map(checkPGUI.class);
        jLabel1.setFont(resourceMap.getFont("jLabel1.font")); // NOI18N
        jLabel1.setText(resourceMap.getString("jLabel1.text")); // NOI18N
        jLabel1.setName("jLabel1"); // NOI18N

        okBtn.setText(resourceMap.getString("okBtn.text")); // NOI18N
        okBtn.setName("okBtn"); // NOI18N
        okBtn.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                okBtnActionPerformed(evt);
            }
        });

        jScrollPane1.setName("jScrollPane1"); // NOI18N

        outAreaTxt.setBackground(resourceMap.getColor("outAreaTxt.background")); // NOI18N
        outAreaTxt.setColumns(20);
        outAreaTxt.setRows(5);

outAreaTxt.setBorder(javax.swing.BorderFactory.createBevelBorder(javax.swing.border.BevelBo
rder.RAISED));
        outAreaTxt.setName("outAreaTxt"); // NOI18N
        jScrollPane1.setViewportView(outAreaTxt);

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addComponent(jLabel1)
                            .addComponent(okBtn)))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(50, 50, 50)
                        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
353, javax.swing.GroupLayout.PREFERRED_SIZE)))
                .addContainerGap(33, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
```

```
                .addGap(33, 33, 33)
                .addComponent(jLabel1)
                .addGap(18, 18, 18)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 203,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 45,
Short.MAX_VALUE)
                .addComponent(okBtn)
                .addContainerGap())
        );

        pack();
    }// </editor-fold>

    private void okBtnActionPerformed(java.awt.event.ActionEvent evt) {
        exitDialog();
    }

    private void formWindowClosed(java.awt.event.WindowEvent evt) {
        exitDialog();
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JButton okBtn;
    private javax.swing.JTextArea outAreaTxt;
    // End of variables declaration
}
```

## A.4 Agent Parameter Group Input Code

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * agentLocSetupGUI.java
 *
 * Created on 27/11/2010, 4:48:44 PM
 */
package ares;

/**
 *
 * @author Paul
 */
import java.io.IOException;
import java.io.RandomAccessFile;
import javax.swing.JOptionPane;
import java.awt.Color;
import java.awt.Graphics;

public class agentLocSetupGUI extends javax.swing.JDialog {

    private String paramFile;
    private String groupName;
    private long hfp;     // pointer to parameter groups header record
    private long gfp;    // pointer to parameter groups data space
    private RandomAccessFile f;  //  @jve:decl-index=0:
    private boolean redActive, blueActive;
    agent redArray[], blueArray[];  // inner class for placement
    int redPtr, bluePtr;
    // parameter group variables
    private int numRedAgents = 0, numBlueAgents = 0;
    private int worldWidth, worldHeight;
    private Graphics g;

    /** Creates new form agentLocSetupGUI */
    public agentLocSetupGUI(java.awt.Frame parent, boolean modal, String file, String tag) {
        super(parent, modal);
        initComponents();

        this.setLocation(200, 200);
        this.setTitle(tag + " Parameters");
        paramFile = file;
        groupName = tag;

        getOtherParameters();

        worldPanelAdditionalSetup();
```

```
        this.setSize(worldWidth + 70, worldHeight + 270);
        redActive = false;
        blueActive = false;
        redPtr = -1;
        bluePtr = -1;
        redArray = new agent[numRedAgents];
        blueArray = new agent[numBlueAgents];
        displayRedCount();
        displayBlueCount();
    }

    // define an inner member class for the agents
    class agent {

        int x, y;
        Color c;

        public void draw(Graphics g) {
            g.setColor(c);
            g.fillOval(x, y, 3, 3);
        }
    }

    void getParameters() {

        boolean exists;
        int i;

        try {
            f = new RandomAccessFile(paramFile, "rwd");

            hfp = AresUtil.findParamGroup(groupName, f);
            if (hfp == 0) {
                JOptionPane.showMessageDialog(this, groupName + " parameter group not found",
"Error Mesage", JOptionPane.ERROR_MESSAGE);
                return;
            }

            f.seek(hfp + 40);
            exists = f.readBoolean();
            gfp = AresUtil.findParamGroupData(groupName, f);

            if (!exists) {
                return;
            }

            // found the group and parameters exist... so read them.
            f.seek(gfp);
            for (i = 0; i < numRedAgents; i++) {
                redPtr++;
                redArray[redPtr] = new agent();
                redArray[redPtr].x = f.readInt();
                redArray[redPtr].y = f.readInt();
                redArray[redPtr].c = Color.red;
                redArray[redPtr].draw(worldPanel.getGraphics());
```

```
            }
            for (i = 0; i < numBlueAgents; i++) {
               bluePtr++;
               blueArray[bluePtr] = new agent();
               blueArray[bluePtr].x = f.readInt();
               blueArray[bluePtr].y = f.readInt();
               blueArray[bluePtr].c = Color.blue;
               blueArray[bluePtr].draw(worldPanel.getGraphics());
            }
            displayRedCount();
            displayBlueCount();
         } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "An error occurred opening this file", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
            exitDialog();
         }
      }

      void getOtherParameters() {
         long gfp;   // pointer to a parameter groups data space

         try {
            f = new RandomAccessFile(paramFile, "rwd");

            gfp = AresUtil.findParamGroupData("Simulation", f);
            if (gfp == 0) {
               JOptionPane.showMessageDialog(null, "Simulation parameter group not found",
"Error Mesage", JOptionPane.ERROR_MESSAGE);
               exitDialog();
            }

            // read the Simulation parameters.
            f.seek(gfp);
            numRedAgents = f.readInt();
            numBlueAgents = f.readInt();

            gfp = AresUtil.findParamGroupData("World", f);
            if (gfp == 0) {
               JOptionPane.showMessageDialog(null, "World parameter group not found", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
               exitDialog();
            }

            // read the World parameters.
            f.seek(gfp);
            worldWidth = f.readInt();
            worldHeight = f.readInt();

         } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "An error occurred opening this file", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
            exitDialog();
         }
      }
```

```
    public void saveParameters() {
        int i;

        if (redPtr < ((numRedAgents - 1))) {
            JOptionPane.showMessageDialog(this, "Not enough Red Agents created", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
            return;
        }
        if (bluePtr < ((numBlueAgents - 1))) {
            JOptionPane.showMessageDialog(this, "Not enough Blue Agents created", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
            return;
        }
        try {

            if (f == null) {
                f = new RandomAccessFile(paramFile, "rwd");
            }

            hfp = AresUtil.findParamGroup(groupName, f);
            if (hfp == 0) {
                JOptionPane.showMessageDialog(this, groupName + " parameter group not found",
"Error Mesage", JOptionPane.ERROR_MESSAGE);
                return;
            }
            gfp = AresUtil.findParamGroupData(groupName, f);


            f.seek(gfp);
            for (i = 0; i < numRedAgents; i++) {
                f.writeInt(redArray[i].x);
                f.writeInt(redArray[i].y);
            }
            for (i = 0; i < numBlueAgents; i++) {
                f.writeInt(blueArray[i].x);
                f.writeInt(blueArray[i].y);
            }
            AresUtil.setParamGroup(groupName, true, f);
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "An error occurred updating the parameter file",
"Error Mesage", JOptionPane.ERROR_MESSAGE);
            exitDialog();
        }
        exitDialog();
    }

    void displayRedCount() {
        redCount.setText("" + (redPtr + 1) + "/" + numRedAgents);
    }

    void displayBlueCount() {
        blueCount.setText("" + (bluePtr + 1) + "/" + numBlueAgents);

    }
```

```
    void clearRed() {
        int i;
        for (i = 0; i <= redPtr; i++) {
            redArray[i].c = Color.white;
            redArray[i].draw(worldPanel.getGraphics());
            redArray[i] = null;
        }
        redPtr = -1;
        displayRedCount();
    }

    void clearBlue() {
        int i;
        for (i = 0; i <= bluePtr; i++) {
            blueArray[i].c = Color.white;
            blueArray[i].draw(worldPanel.getGraphics());
            blueArray[i] = null;
        }
        bluePtr = -1;
        displayBlueCount();
    }

    public void exitDialog() {
        try {
            f.close();
        } catch (IOException e) {
            JOptionPane.showMessageDialog(this, "An error occurred closing the parameter file",
"Error Mesage", JOptionPane.ERROR_MESSAGE);
            exitDialog();
        }
        setVisible(false);
        dispose();
    }

    public void worldPanelAdditionalSetup() {
        worldPanel.setSize(worldWidth+10, worldHeight+10);
        worldPanel.addMouseListener(new java.awt.event.MouseAdapter() {

            public void mouseReleased(java.awt.event.MouseEvent e) {
                if (redActive) {
                    if (redPtr < (numRedAgents - 1)) {
                        redPtr++;
                        redArray[redPtr] = new agent();
                        redArray[redPtr].x = e.getX();
                        redArray[redPtr].y = e.getY();
                        redArray[redPtr].c = Color.red;
                        redArray[redPtr].draw(worldPanel.getGraphics());
                        msg.setText("x = " + redArray[redPtr].x + "  y - " + redArray[redPtr].y);
                        displayRedCount();
                    } else {
                        msg.setText("No more red agents to allocate");
                        redActive = !redActive;
                    }
                }
                if (blueActive) {
```

```
                if (bluePtr < (numBlueAgents - 1)) {
                    bluePtr++;
                    blueArray[bluePtr] = new agent();
                    blueArray[bluePtr].x = e.getX();
                    blueArray[bluePtr].y = e.getY();
                    blueArray[bluePtr].c = Color.blue;
                    blueArray[bluePtr].draw(worldPanel.getGraphics());
                    msg.setText("x = " + blueArray[bluePtr].x + "  y - " + blueArray[bluePtr].y);
                    displayBlueCount();
                } else {
                    msg.setText("No more blue agents to allocate");
                    blueActive = !blueActive;
                }
            }
        }
    });

}

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jLabel1 = new javax.swing.JLabel();
    addRedBtn = new javax.swing.JButton();
    addBlueBtn = new javax.swing.JButton();
    redCount = new javax.swing.JLabel();
    blueCount = new javax.swing.JLabel();
    clearRedBtn = new javax.swing.JButton();
    clearBlueBtn = new javax.swing.JButton();
    saveBtn = new javax.swing.JButton();
    cancelBtn = new javax.swing.JButton();
    msg = new javax.swing.JTextField();
    loadBtn = new javax.swing.JButton();
    worldPanel = new javax.swing.JPanel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
    setLocationByPlatform(true);
    setName("Form"); // NOI18N
    getContentPane().setLayout(null);

    org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(ares.AresApp.class).getContext().getResource
Map(agentLocSetupGUI.class);
    jLabel1.setFont(resourceMap.getFont("jLabel1.font")); // NOI18N
    jLabel1.setText(resourceMap.getString("jLabel1.text")); // NOI18N
    jLabel1.setName("jLabel1"); // NOI18N
    getContentPane().add(jLabel1);
```

```
jLabel1.setBounds(70, 20, 234, 29);

addRedBtn.setText(resourceMap.getString("addRedBtn.text")); // NOI18N
addRedBtn.setName("addRedBtn"); // NOI18N
addRedBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        addRedBtnActionPerformed(evt);
    }
});
getContentPane().add(addRedBtn);
addRedBtn.setBounds(37, 87, 73, 23);

addBlueBtn.setText(resourceMap.getString("addBlueBtn.text")); // NOI18N
addBlueBtn.setName("addBlueBtn"); // NOI18N
addBlueBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        addBlueBtnActionPerformed(evt);
    }
});
getContentPane().add(addBlueBtn);
addBlueBtn.setBounds(206, 87, 73, 23);

redCount.setText(resourceMap.getString("redCount.text")); // NOI18N
redCount.setName("redCount"); // NOI18N
getContentPane().add(redCount);
redCount.setBounds(90, 121, 41, 14);

blueCount.setText(resourceMap.getString("blueCount.text")); // NOI18N
blueCount.setName("blueCount"); // NOI18N
getContentPane().add(blueCount);
blueCount.setBounds(259, 121, 45, 14);

clearRedBtn.setText(resourceMap.getString("clearRedBtn.text")); // NOI18N
clearRedBtn.setName("clearRedBtn"); // NOI18N
clearRedBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        clearRedBtnActionPerformed(evt);
    }
});
getContentPane().add(clearRedBtn);
clearRedBtn.setBounds(37, 146, 77, 23);

clearBlueBtn.setText(resourceMap.getString("clearBlueBtn.text")); // NOI18N
clearBlueBtn.setName("clearBlueBtn"); // NOI18N
clearBlueBtn.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        clearBlueBtnActionPerformed(evt);
    }
});
getContentPane().add(clearBlueBtn);
clearBlueBtn.setBounds(206, 146, 79, 23);

saveBtn.setText(resourceMap.getString("saveBtn.text")); // NOI18N
saveBtn.setName("saveBtn"); // NOI18N
saveBtn.addActionListener(new java.awt.event.ActionListener() {
```

```
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            saveBtnActionPerformed(evt);
        }
    });
    getContentPane().add(saveBtn);
    saveBtn.setBounds(410, 109, 57, 23);

    cancelBtn.setText(resourceMap.getString("cancelBtn.text")); // NOI18N
    cancelBtn.setName("cancelBtn"); // NOI18N
    cancelBtn.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            cancelBtnActionPerformed(evt);
        }
    });
    getContentPane().add(cancelBtn);
    cancelBtn.setBounds(410, 143, 65, 23);

    msg.setText(resourceMap.getString("msg.text")); // NOI18N
    msg.setName("msg"); // NOI18N
    getContentPane().add(msg);
    msg.setBounds(37, 180, 346, 20);

    loadBtn.setText(resourceMap.getString("loadBtn.text")); // NOI18N
    loadBtn.setName("loadBtn"); // NOI18N
    loadBtn.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            loadBtnActionPerformed(evt);
        }
    });
    getContentPane().add(loadBtn);
    loadBtn.setBounds(410, 177, 55, 23);

    worldPanel.setBackground(resourceMap.getColor("worldPanel.background")); // NOI18N
    worldPanel.setBorder(javax.swing.BorderFactory.createEtchedBorder());
    worldPanel.setMaximumSize(new java.awt.Dimension(65536, 65536));
    worldPanel.setName("worldPanel"); // NOI18N
    worldPanel.setLayout(null);
    getContentPane().add(worldPanel);
    worldPanel.setBounds(18, 211, 457, 160);

    jLabel2.setText(resourceMap.getString("jLabel2.text")); // NOI18N
    jLabel2.setName("jLabel2"); // NOI18N
    getContentPane().add(jLabel2);
    jLabel2.setBounds(37, 121, 43, 14);

    jLabel3.setText(resourceMap.getString("jLabel3.text")); // NOI18N
    jLabel3.setName("jLabel3"); // NOI18N
    getContentPane().add(jLabel3);
    jLabel3.setBounds(206, 121, 43, 14);

    pack();
}// </editor-fold>

private void saveBtnActionPerformed(java.awt.event.ActionEvent evt) {
    saveParameters();
```

```
    }

    private void cancelBtnActionPerformed(java.awt.event.ActionEvent evt) {
        exitDialog();
    }

    private void addRedBtnActionPerformed(java.awt.event.ActionEvent evt) {
        blueActive = false;
        redActive = !redActive;
    }

    private void addBlueBtnActionPerformed(java.awt.event.ActionEvent evt) {
        redActive = false;
        blueActive = !blueActive;
    }

    private void clearRedBtnActionPerformed(java.awt.event.ActionEvent evt) {
        clearRed();
    }

    private void clearBlueBtnActionPerformed(java.awt.event.ActionEvent evt) {
        clearBlue();
    }

    private void loadBtnActionPerformed(java.awt.event.ActionEvent evt) {
        loadBtn.setEnabled(false);
        loadBtn.setVisible(false);
        clearRed();
        clearBlue();
        getParameters();
    }
    // Variables declaration - do not modify
    private javax.swing.JButton addBlueBtn;
    private javax.swing.JButton addRedBtn;
    private javax.swing.JLabel blueCount;
    private javax.swing.JButton cancelBtn;
    private javax.swing.JButton clearBlueBtn;
    private javax.swing.JButton clearRedBtn;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JButton loadBtn;
    private javax.swing.JTextField msg;
    private javax.swing.JLabel redCount;
    private javax.swing.JButton saveBtn;
    private javax.swing.JPanel worldPanel;
    // End of variables declaration
}
```

## A.5 Simulation Engine Code Listing

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package ares;

/**
 *
 * @author Paul
 */
import java.io.IOException;
import java.io.RandomAccessFile;
import java.lang.Thread;
import java.util.Vector;
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class SimEngine  implements Runnable {
   String paramFile;  // name of the parameter file from which the simulation is initialized
   JFrame gui;        // reference to the original Ares GUI so it can be made vbisible when
finished

   int simRuns;            // number of times to run the simulation
   int maxTimeSteps;       // number of time steps to run the simulation
   int worldHeight, worldWidth; // world size elements

   SimulationStatusBar bar;
   SimManager simMan;
   ActionManager actMan;


   /*************************************************************************/

   // Constructor
   SimEngine(String pf, JFrame g) {
      gui = g;
      paramFile = pf;
      bar = new SimulationStatusBar();
      bar.setVisible(true);
   }


   public void run() {

      // initialize all the required elements from the passed parameter file
      initializeSimulation();
      simMan = new SimManager(paramFile);  // create the Sims Manager object
      actMan = new ActionManager();        // create the Action Manager object
      runSimulation();
      exitSimEngine();
```

```
  }


  /*
   * This is the method that actually runs the simulation.
   *
   * The simulation is run a number of times, depending on the value of the simRuns variable.
    *At each iteration the simulation is re-started with exactly the same parameters, and runs
    * till the iteration finishes.  Each iteration finishes when all agents from an opposing team
    * are eliminated, or a predefined number of time steps has been reached - the value of
    * maxTimeSteps.
   *paul D
   */

  void runSimulation() {
     int currRun, time;
     boolean finishedRun;

     for (currRun = 1; currRun <= simRuns; currRun++) {  // iterate through all the simulation
runs
          // need to reset the simulation each time it re-starts ???
        time = 1;
        finishedRun = false;
        bar.setRun(currRun);

        while (!finishedRun) {  // now perform the current simulation run
          bar.setTime(time);
          simMan.cognition();       // all agents do their thinking and assemble their actions
          actMan.processActions();  // now process all the sims actions

          time++;
          finishedRun = isFinished(time);
        }

     }
  }

  void exitSimEngine() {
              bar.dispose();
              gui.setVisible(true);
  }

         // this method checks to see if the simulation run has finished
         boolean isFinished (int time) {
           if (maxTimeSteps > 0) {
             if (time > maxTimeSteps) return true;
           }

           if (bar.getPauseStatus()) pauseMode();

           return false;
  }

  void pauseMode() {
    while (bar.getPauseStatus() ) {
```

```
          //System.out.println("I'm sleeping");
          try {
          Thread.sleep(1000);
          } catch (Exception Ex) {}
    }
  }

  void initializeSimulation() {
            RandomAccessFile f;
            long gfp;   // pointer to a parameter groups data space
            int i, x, y, agentNo;

      try {
          f = new RandomAccessFile(paramFile,"rwd");

          gfp = AresUtil.findParamGroupData("Simulation", f);
          if (gfp == 0 ) {
                    JOptionPane.showMessageDialog(null,"Simulation parameter group not
found","Error Mesage",JOptionPane.ERROR_MESSAGE);
                    exitSimEngine();
          }

          // read the Simulation parameters.
          f.seek(gfp);
          f.readInt();
          f.readInt();
          simRuns = f.readInt();
          maxTimeSteps = f.readInt();

          gfp = AresUtil.findParamGroupData("World", f);
          if (gfp == 0 ) {
                    JOptionPane.showMessageDialog(null,"World parameter group not found","Error
Mesage",JOptionPane.ERROR_MESSAGE);
                    exitSimEngine();
          }

          // read the World parameters.
          f.seek(gfp);
          worldWidth = f.readInt();
          worldHeight = f.readInt();

             // close the parameter file
             f.close();
        }
       catch (IOException e) {
          JOptionPane.showMessageDialog(null,"An error occurred opening this file","Error
Mesage",JOptionPane.ERROR_MESSAGE);
          exitSimEngine();
       }
  }

}
```

## A.6 Sim Manager Code Listing

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ares;

import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;

import javax.swing.JOptionPane;

/**************************************************************************
 * The SimManager class is a class created by the SimEngine and its purpose is to control
 * and manage all the Sims throughout the simulation. This class allievates the SimEngine
 * class from having to deal with the micro-management of the sims and provides methods for
 * controlling the thinking pattern during the simulation and handling deaths, and well as
 * letting the SimEngine know when all Sims in one team have been eliminated.
 *
 * Paul D
 *
 * Created on Jul 2006
 *************************************************************************/
public class SimManager {

    String paramFile;  // name of the parameter file from which the agents are initialized
    int numBlueAgents, numRedAgents;  // number of agents
    // Variables concerned with the agents
    Sim agents[];

    SimManager(String pf) {
        paramFile = pf;
        initializeSims();

    }

    // the cognition method is called from the SimEngine object and gets all agents to do their
thinking
    void cognition() {
        int i;
        for (i = 0; i < agents.length; i++) {
            agents[i].mind.think();
        }

    }

    void exitSimManager() {
    }

    Sim[] getDetectedSims(Sim a) {
        int i;
```

```
        double dx, dy, distance;
        ArrayList tmp;
        Sim ret[];
        String s;

        //System.out.println("I am agent "+a.agentID+" (team "+a.team+") at location
"+a.loc.x+","+a.loc.y);
        //System.out.println();
        tmp = new ArrayList();
        for (i = 0; i < agents.length; i++) {
            if (agents[i].agentID == a.agentID) {
                continue;
            }
            if (agents[i].alive) {
                dx = a.loc.x - agents[i].loc.x;
                dy = a.loc.y - agents[i].loc.y;
                distance = Math.sqrt((dx * dx) + (dy * dy));
                if (distance <= a.visionRange) {
                    tmp.add(agents[i]);
                    //System.out.println("Found agent "+agents[i].agentID+" (team
"+agents[i].team+") at location "+agents[i].loc.x+","+agents[i].loc.y);
                }
            }
        }
        //System.out.println("found "+tmp.size()+" agents");
        //System.out.println();
        //s = ioUtil.getString();
        //now create and return the array of detected agents
        //JOptionPane.showMessageDialog(null,"Sim "+a.agentID+" detected "+tmp.size()
+"other sims","Error Mesage",JOptionPane.INFORMATION_MESSAGE);

        ret = new Sim[tmp.size()];
        tmp.toArray(ret);
        return ret;
    }

    private void initializeSims() {
        RandomAccessFile f;
        long gfp;   // pointer to a parameter groups data space
        int i, x, y, agentNo;

        try {
            f = new RandomAccessFile(paramFile, "rwd");

            gfp = AresUtil.findParamGroupData("Simulation", f);
            if (gfp == 0) {
                JOptionPane.showMessageDialog(null, "Simulation parameter group not found",
"Error Mesage", JOptionPane.ERROR_MESSAGE);
                exitSimManager();
            }

            // read the Simulation parameters.
            f.seek(gfp);
            numRedAgents = f.readInt();
            numBlueAgents = f.readInt();
```

```
            gfp = AresUtil.findParamGroupData("Agents", f);
            if (gfp == 0) {
               JOptionPane.showMessageDialog(null, "Agents parameter group not found", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
               exitSimManager();
            }
            f.seek(gfp);

            //         read the agent positions and build the agent array
            agentNo = 0;
            agents = new Sim[numRedAgents + numBlueAgents];

            // first the red sims - team 0
            for (i = 0; i < numRedAgents; i++) {
               x = f.readInt();
               y = f.readInt();
               // System.out.println("agent "+agentNo+"  at location "+x+","+y);
               agents[agentNo] = new Sim(agentNo, 0, x, y, "RedMind", this);
               agentNo++;
            }
            // now the blue sims - team 1
            for (i = 0; i < numBlueAgents; i++) {
               x = f.readInt();
               y = f.readInt();
               // System.out.println("agent "+agentNo+"  at location "+x+","+y);
               agents[agentNo] = new Sim(agentNo, 1, x, y, "BlueMind", this);
               agentNo++;
            }

            // close the parameter file
            f.close();
         } catch (IOException e) {
            JOptionPane.showMessageDialog(null, "An error occurred opening this file", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
            exitSimManager();
         }
      }
}
```

## A.7 Sim Class Code Listing

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package ares;

import javax.swing.text.Position;
import javax.swing.JOptionPane;
import java.lang.reflect.*;

/**********************************************************************
 * The Sim class represents the physical aspects of simulated agents in the
 * world.  It contains all things relevant to an agents physical location,
 * health, well being etc...  Each simulated agent has two aspects... a Sim
 * object (its physical side) and a SimMind object (representing its cognitive
 * side).
 *
 * Paul D
 *
 * Created on Jan 2006
 **********************************************************************/
public class Sim {

    double size; // the radius of this agent in meters
    double mass; // the mass of this agent in kilograms
    boolean alive; // is the agent alive
    int health;    // the health of the agent, specified as an integer representing a number of
health points
    int maxHealth; // the maximum health points of the agent
    int agentID;  // the individual agent code - this is unique for each agent
    int team;     // an integer indicating the team the agent belongs to: currently Red - 0, Blue -
1
    Location loc;        // the current location of the agent within the world
    double facing;       // the direction the agent is facing
    double visionRange;  // max range of vision for this agent
    double maxSpeed;     // max speed the agent can move at
    double weaponRange;  // max range of the current default weapon
    Senses sense;     // the object which will provide access to the Agents sensing capability
    SimManager sman;  // reference to the Sim Manager object (used by sense object)
    SimMind mind;     // the mind controlling the cognitive aspects of this agent

    Sim(int id, int t, int x, int y, String m, SimManager sm) {
        agentID = id;
        team = t;
        loc = new Location(x, y);
        alive = true;
        health = 100;
        maxHealth = 100;

        visionRange = 250.0;
        maxSpeed = 4.0;
```

```
        weaponRange = 120.0;

        sman = sm;
        sense = new Senses(this);
        //JOptionPane.showMessageDialog(null, "Mind Name : "+m+" ", "Error Mesage",
JOptionPane.ERROR_MESSAGE);

        //instantiate the agents cognitave component
        try {
            //Class c = Class.forName(m,true,ClassLoader.getSystemClassLoader());
            //Class c = Class.forName(m);
            //Thread th = Thread.currentThread();
            //ClassLoader cl = th.getContextClassLoader();
            //Class c = cl.loadClass(m);
            if (m.contentEquals("RedMind")) {
                mind = new RedMind();
            }
            else {
                if (m.contentEquals("BlueMind")) {
                    mind = new BlueMind();
                } else {
                    JOptionPane.showMessageDialog(null, "Invalid mind object found", "Error
Mesage", JOptionPane.ERROR_MESSAGE);
                }
            }
            //mind = (SimMind) c.newInstance();
            mind.setBody(this);
        } catch (Exception e) { /* need to handle this exception */

        }
    }
}
```

## A.8 SimMind Object Code Listing

```
package ares;

/**************************************************************************
 * This class is the abstract class that represents the cognitive aspect of physical
 * agent represented by the 'Sim' class. The Sim class represents physical component
 * of the agent, and all agents have the same physical component.  Different agents
 * with different cognitive aspects, or 'control paradigms' will have a different
 * 'mind' component which extends 'SimMind'
 *
 * Paul D
 ************************************************************************** */
public abstract class SimMind {

   Sim body;  // the body variable will hold a reference to the physical aspect of the agent... its
body

   SimMind() {
   }

   void setBody(Sim x) {
      body = x;
   }

   /*
    This is the method that is called for each agent for each time-step in the simulation.  This
    method represents the thinking process that the agent undergoes before deciding on a
    course of action for this time-step.  Each agents think() nmethod is allowed to run to
    completion, before the actions decided upon by each of the agents is applied ... in a
    random order
    */

   public abstract void think();
   }
```

## A.9 Action Manager Code Listing

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package ares;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;


/**************************************************************************
 * The ActionManager class is a class created by the SimEngine and its purpose is to control
 * and manage all the actions created by the various Sims during their thinking phase. The
 * processing of the actions is initiated by the SimEngine object by calling the
 * ActionManagers processActions() methods... which in turn calls each of the actions
 * process() method.
 *
 * To make sure the simulation is as unbiased as possible, the list of actions to be processed
 * is randomized before processing.
 *
 * Paul D
 *
 * Created on Aug 2006
 *************************************************************************/

public class ActionManager {

        Random mixer;   // used as the source to shuffle the action list
        // Variables concerned with the actions
        ArrayList<Action> actions;   // this will contain all the actions

        ActionManager() {
                actions = new ArrayList();
                mixer = new Random();
                actions.clear();
        }

        void processActions() {
                int i;
                Action a[];

                // first randomize the actions
                Collections.shuffle(actions,mixer);

                // make an arry from the actionlist
                a = new Action[actions.size()];
                actions.toArray(a);

                // now process all the actions
```

```
                for (i = 0; i < a.length; i++) {
                        if (a[i].owner.alive) {
                                a[i].process();
                        }
                }

                // now clear out the actions ready for a new round
                actions.clear();
        }

        void exitActionManager() {
        }

}
```

## A.10 Action Object Code Listing

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package ares;

/**************************************************************************
 * This class is the abstract class that all actions must inherit from. It defines
 * an owner of the action, along with an abstract method called process() that all
 * concrete action classes must implement.
 *
 * The actions are managed my the ActionManager class, created by the SimEngine
 * object
 *
 * Paul D
************************************************************************** */


public abstract class Action {

        Sim owner;   // the owner of the action

          /*
           * This is the method that is called for each of the actions by the ActionManager
           * class when processing all the actions at the end of the simulation turn. This
method
           * must be implemented by all concrete action classes, and 'pplies' the action they
           * represent.
           */

        public abstract void process();


}
```

## A.11 BlueMind Code Listing

```
package ares;

/*
   This is a very basic Agent which represents an extension of the SimMind class for use
   in the Ares simulation.  This class employs  Reinforcement Learning  concepts for a
   comparison to the Instinctual agents whose behaviour is goverened by
   behavioural triggers or springs as outlined in the Crocadile manual.

   This represents the third version... that now implements communication
   between team members to allow the communication of rewards to each other.
   This was implemented to try and speen up the learning process.

   Also, this version stores State-Action-Time tupples in a vector because it
   was found that rewards were being allocated against actions & states that
   were not responsible for them... due to time delay in munitions.  The rewards
   are passed to this agent in a vector from the accoriated Agent,and have a
   time attached to them, so the appropriate state-action-time tupple
   corresponding to the reward can be used to update the Q* values correctly.

   Author: Paul Darbyshire
   Version 3
   18/6/2002 Added comms for communication of reward
*/

import java.io.*;
import java.util.StringTokenizer;
import java.util.Vector;

public class BlueMind extends SimMind {

   CapabilityManager    capMan;
   InformationManager   infMan;
   StatusMonitor        stsMon;

   // Q-Learning Learning Structure to hold state-action values for
   RLStateAction q[];

   Vector sat;  // small vector to hold State-Action-Time tupples

   //Some Reinforcement Learning constants
   private final double stepSize = 0.1;
   private final double discountRate = 0.0;
   private final double e = 0.1;  // probability used for e-greedy exploration

   private final int SATLENGTH = 20; // the length of the sat vector
   private final double pi = 3.14159;    // PI

   double dir;       // just set a random direction

   // needed runtime variables
   int oldState = -1, newState = -1; // to keep track of the old and new states
```

```
  int oldAction, newAction;         // keeps track of the old and new actions
  double reward;

  public BlueMind(InformationManager information, CapabilityManager capability, StatusMonitor
statMon, SimMindSetup abs) {
     super(information, capability, statMon, abs);
     setup(abs);
          }

  /*
     This is an implementation of the abstract method defined in AgentBrain
     Its purpose is to provide a point for initialization before the simulation begins
  */
  public void setup(SimMindSetup setup) {

      capMan = super.getCaps();
      infMan = super.getInfo();
      stsMon = super.getStatus();
      dir = stsMon.getDirectionFacing();
      //dir = RandomGen.nextInt(4)*pi/2;

      // initialize the sat vector
      sat = new Vector(SATLENGTH+2);

      // now build the q array representing the action-value approximation Q(s,a)
      try {
         q = RLUtil.buildQ();
         //System.out.println("length of q array is: "+q.length);
      } catch (IOException e) {System.out.println("Somethings happened");}
  }

  /*-------------------------------------------------------------------------*/

  /*
     This is an implementation of the abstract method defined in SimMind

     This is the method that is called in each round of the simulation to determine the
     associated physical agents behaviour and actions
  */
  public void think() {
   Vector rewards;
   RLReward r;
   RLSATtupple t;
   int i;

   scan();
   oldState = newState;
   oldAction = newAction;
   newState = getCurrentState();
   reward = 0;

   // get the rewards vector and then process them all
   rewards = stsMon.getRewards();
   for (i = 0; i < rewards.size(); i++) {
      r = (RLReward) rewards.elementAt(i);
```

```
      reward += r.reward;  // look into getting rid of this
      System.out.println((int)infMan.getSimTime()+": "+stsMon.getID()+": found reward =
"+r.reward+"  from time "+r.time);

      // now find the state-and action that generated it - scan through sat
      t = getSAT(r.time);
      if (t != null) {
         if (r.reward > 0) sendComms(t.state, t.action, r.reward, newState);

         // now procedss this reward
         processReward(t.state, t.action, r.reward, newState);
      }
      else
         System.out.println("Couldn't find corresponding SAT object");
   }

//    if (oldState != -1) { // not first time around
//       processReward(oldState, oldAction, reward, newState);
//    }

    // check comms for team members experience before choosing an action
    receiveComms();

    newAction = chooseNewAction(newState);

    // now store the current state, action and simulation time for later
    if (sat.size() >= SATLENGTH) sat.removeElementAt(0); // manitains size
    sat.addElement(new RLSATtupple(newState, newAction, infMan.getSimTime()));

    takeAction(newState, newAction);
   }

   /*-------------------------------------------------------------------------*/

   /*  This method processes the reward received from the system.  Currently
       uses equation 6.6 from Sutton & Barto for Q-Learning

       required parameters
       state  - state of system when action was chosen
       action - action chosen that leads to this reward
       r      - the reward gained from State- Action pair
       nstate - the new state resulting from the application of the action
   */
   private void processReward(int state, int action, double r, int nstate) {

     // update the action-value matrix
     q[state].values[action] = q[state].values[action] +
            stepSize * (r + discountRate*maxActions(q[nstate]) - q[state].values[action]);
   }

   /*-------------------------------------------------------------------------*/

   /*  Just construct a simple string message with the following elements
      State  - state agent was in corresponding to the reward obtained
      Action - action chosen in the given state leading to the reward
```

```
    Reward - value of reward received
    sender - who's sending the message
    nstate - the new state (current new state)
  */
  private void sendComms(int state, int action, double reward, int nstate) {
    String msg;

    msg = "State "+state+"  Action "+action+"  Reward "+reward+"  newState "+nstate+"
from "+stsMon.getID();
    capMan.getCurrentComms().send(msg);
  }

  /*-------------------------------------------------------------------------*/

  /* This method retrieves generic messages sent as strings from team members
    and decodes them.  At this stage, the only generic message should be
    reward discoveries sent by team members
  */
  private void receiveComms() {
    GenericMessage msgs[];
    int i;
    StringTokenizer st;
    String s;

    // variables to hold the decoded message information
    int state, action, sender, nstate;
    double rw;   // reward

    msgs = infMan.getGenericMessages();
    for (i = 0; i < infMan.getNoOfGenericMessages(); i++) {
        // decode the message string first
        st = new StringTokenizer(msgs[i].getContent());
        if (st.countTokens() != 10) continue;   // means string is not the right type
        s = st.nextToken(); s = st.nextToken();
        state = Integer.parseInt(s);
        s = st.nextToken(); s = st.nextToken();
        action = Integer.parseInt(s);
        s = st.nextToken(); s = st.nextToken();
        rw = Double.parseDouble(s);
        s = st.nextToken(); s = st.nextToken();
        nstate = Integer.parseInt(s);
        s = st.nextToken(); s = st.nextToken();
        sender = Integer.parseInt(s);

      System.out.print("   "+(int)infMan.getSimTime()+": "+stsMon.getID()+" -
"+msgs[i].getContent());
      if (sender  != stsMon.getID()) {
        System.out.print("   "+state+" "+action+" "+rw+" "+nstate+" "+sender);
      }
      System.out.println();

      processReward(state, action, rw, nstate);
    }
  }
```

```
/*-----------------------------------------------------------------------*/

// returns the sat element corresponding to the passed time parameter

private RLSATtupple getSAT(double time) {
   int i;
   for (i = sat.size()-1; i >= 0; i--) {
      if (((RLSATtupple)sat.elementAt(i)).time == time) {
         return (RLSATtupple)sat.elementAt(i);
      }
   }
   return null;  // not found
}

/*-----------------------------------------------------------------------*/

private int chooseNewAction(int s) {
   double max;
   int actionIndex, i,j;
   int indexes[];

   // first see if we will do an e-greedy exploration
   if (RandomGen.getRand() < e) {
      // choose a random action from this state
      actionIndex = RandomGen.nextInt(q[s].values.length);
      return actionIndex;
   }

   // else now find the maximum action-value
   actionIndex = 0;
   max = q[s].values[actionIndex];
   for (i = 0; i < q[s].values.length; i++) {
      if (q[s].values[i] > max) {
         max = q[s].values[i];
         actionIndex = i;
      }
   }

   // now see if there is more than one of this value
   indexes = new int[q[s].values.length];
   j = 0;
   for (i = 0; i < q[s].values.length; i++) {
      if (q[s].values[i] == max) {
         indexes[j] = i;
         j++;
      }
   }
   // if there is then randomly choose one of them
   if (j > 1) actionIndex = indexes[RandomGen.nextInt(j)];

   return actionIndex;
}

/*-----------------------------------------------------------------------*/
```

```
private void takeAction(int s, int a) {
   if (q[s].actions[a].equals("nothing")) {
         // do nothing for a simulation turn
      doNothing();
   }
   else if (q[s].actions[a].equals("movaway")) {
         // move away from the locus point of all enemy
      moveAwayFromEnemy();
   }
   else if (q[s].actions[a].equals("movto")) {
         // moves to the closest enemy agent
      moveToEnemy();
   }
   else if (q[s].actions[a].equals("fire")) {
         // fires at the first enemy in the Line Of Sight
      fireAtTargets();
   }
   else if (q[s].actions[a].equals("movtoF")) {
         // moves to the closest enemy and fires at first in LOS
         fireAtTargets();
         moveToEnemy();
   }
   else if (q[s].actions[a].equals("movtoW")) {
      // moves to the nearest wounded enemy
         moveToWounded();
   }
   else if (q[s].actions[a].equals("fireW")) {
      // fires at the first wounded in the LOS
         fireAtWounded();
   }
   else if (q[s].actions[a].equals("movtoFW")) {
      // moves to the closest enemy and fires at first wounded in LOS
         fireAtWounded();
         moveToEnemy();
   }
   else if (q[s].actions[a].equals("movawayF")) {
      // move away from the locus point of all enemy and fire at closest in LOS
         fireAtTargets();
      moveAwayFromEnemy();
   }
   else if (q[s].actions[a].equals("movawayFW")) {
      // move away from the locus point of all enemy and fire at closest wounded in LOS
         fireAtWounded();
      moveAwayFromEnemy();
   }
   else if (q[s].actions[a].equals("explore")) {
         explore();
   }
   else {
         System.out.println("UNKNOWN action discovered in BasicRLAgent takeAction()");
         System.exit(0);
   }
}

/*-------------------------------------------------------------------------*/
```

```
// this represents a move in a random direction
private void explore() {
   Movement m;

   m = capMan.getCurrentMovement();

   // only change direction if a negative reward
   if (reward < 0) dir = RandomGen.nextInt(4)*pi/2;

   m.setDirection(dir);
   m.setSpeed(m.getMaxSpeed()/2);
   m.applyChanges();

}

/*-------------------------------------------------------------------------*/

private void doNothing() {
   capMan.getCurrentMovement().setSpeed(0);
   capMan.getCurrentMovement().applyChanges();
}

/*-------------------------------------------------------------------------*/

private void moveAwayFromEnemy() {

   DetectedAgent en[];
   int i;
   Position locus;
   double locusZ, locusX;
   Vect3D vectToLocus;
   Movement m;

   if (infMan.getNoOfEnemies() == 0) {
      doNothing();
      return;
   }
   en = infMan.getEnemies();

   // now get the locus point of all the detected agents
   locusZ = 0; locusX = 0;
   for (i = 0; i < en.length; i++) {
      locusX = locusX + en[i].getPosition().getX();
      locusZ = locusZ + en[i].getPosition().getZ();
   }
   locusX = locusX / en.length;
   locusZ = locusZ / en.length;
   // now create a Position from the locus and get a vector to it
   locus = new Position(locusX, locusZ);
   vectToLocus = stsMon.getVectorTo(locus);

   // now set a movement away from the locus point at max speed
   m = capMan.getCurrentMovement();
   vectToLocus.inverse();  // invert so direction is away
```

```
    m.adjustForVelocity(vectToLocus);
    m.setSpeed(m.getMaxSpeed());
    m.applyChanges();

  }

  /*-------------------------------------------------------------------------*/

  private void moveToEnemy() {

    DetectedAgent en[];
    double dist;
    int i, index;
    Movement m;

    if (infMan.getNoOfEnemies() == 0) {
      doNothing();
      return;
    }

    en = infMan.getEnemies();
    if (en.length <= 0) return;

    // otherwise find the closest, then move towards them
    index = 0;
    dist = stsMon.getPosition().distanceTo(en[0].getPosition());
    for (i = 0; i < en.length; i++)
      if (stsMon.getPosition().distanceTo(en[i].getPosition()) < dist) {
        dist = stsMon.getPosition().distanceTo(en[i].getPosition());
        index = i;
      }
    m = capMan.getCurrentMovement();
    m.setDirection(Position.getDirectionFromBToA(en[index].getPosition(),
stsMon.getPosition()));
    m.setSpeed(m.getMaxSpeed()/2);
    m.applyChanges();
  }

  /*-------------------------------------------------------------------------*/

  private void moveToWounded() {
    DetectedAgent en[];
    double dist;
    int i, index;
    Movement m;

    en = infMan.getEnemies();
    if (en.length <= 0) return;

    // otherwise find the closest wounded, then move towards them
    index = 0;
    dist = stsMon.getPosition().distanceTo(en[0].getPosition());
    for (i = 0; i < en.length; i++)
      if ((en[i].getHealth() < en[i].getMaxHealth()) &&
(stsMon.getPosition().distanceTo(en[i].getPosition()) <= dist)) {
```

```
          dist = stsMon.getPosition().distanceTo(en[i].getPosition());
          index = i;
        }
      m = capMan.getCurrentMovement();
      m.setDirection(Position.getDirectionFromBToA(en[index].getPosition(),
stsMon.getPosition()));
      m.setSpeed(m.getMaxSpeed()/2);
      m.applyChanges();
    }

    /*-------------------------------------------------------------------------*/

    private void fireAtTargets() {

      DetectedAgent en[];
      int i;

      if (infMan.getNoOfEnemies() == 0) {
        doNothing();
        return;
      }

      en = getEnemyInRange();     // get all the enemies in sight
      if (en == null) return;

      // if we can't fire then return
      if (capMan.getCurrentWeapon() == null) return;

      //otherwise - fire at first enemy in line of sight
      for (i = 0; i < en.length; i++) {
        if (infMan.getTerrain().isLineOfSight(stsMon.getPosition(), en[i].getPosition())) {
          capMan.getCurrentWeapon().fire(en[i]);
          return;
        }
      }
    }

    /*-------------------------------------------------------------------------*/

    private void fireAtWounded() {

      DetectedAgent en[], w[];
      int i;

      // if we can't fire then return
      if (capMan.getCurrentWeapon() == null) return;

      w = getWoundedInRange();
      if (w == null) return;

      //otherwise - fire at first wounded enemy in line of sight
      for (i = 0; i < w.length; i++) {
        if (infMan.getTerrain().isLineOfSight(stsMon.getPosition(), w[i].getPosition())) {
          capMan.getCurrentWeapon().fire(w[i]);
          return;
```

```
      }
    }

    // if none in sight... then fire at first enemy in line of sight
    en = getEnemyInRange();     // get all the enemies in sight
    if (en == null) return;
    for (i = 0; i < en.length; i++) {
      if (infMan.getTerrain().isLineOfSight(stsMon.getPosition(), en[i].getPosition())) {
        capMan.getCurrentWeapon().fire(en[i]);
        return;
      }
    }
  }

  /*-------------------------------------------------------------------------*/

  private DetectedAgent[] getWoundedInRange() {
          DetectedAgent en[], temp[];
          int i,j, wounded = 0;

          en = getEnemyInRange();

          // find out how many of them are wounded
    for (i = 0; i < en.length; i++) if (en[i].getHealth() < en[i].getMaxHealth()) wounded++;

    // if none are wounded, return null otherwise construct the array of wounded enemies
          if (wounded == 0) return null;
          temp = new DetectedAgent[wounded];
          j = 0;
    for (i = 0; i < en.length; i++)
      if (en[i].getHealth() < en[i].getMaxHealth()) {
        temp[j] = en[i];
        j++;
      }

    // the array should be sorted so just return it
    return temp;
  }

  /*-------------------------------------------------------------------------*/

  private DetectedAgent[] getEnemyInRange() {
    DetectedAgent en[], temp[], t;
    double weaponRange;
    int i, j, swaps, inRange = 0;

    weaponRange = capMan.getCurrentWeapon().getMaxRange();
    en = infMan.getEnemies();

    // find out how many in range
    for (i = 0; i < en.length; i++) if (stsMon.getPosition().distanceTo(en[i].getPosition()) <=
weaponRange) inRange++;

    // if none in range, return null otherwise construct the array of enemies
          if (inRange == 0) return null;
```

```
            temp = new DetectedAgent[inRange];
            j = 0;
      for (i = 0; i < en.length; i++)
        if (stsMon.getPosition().distanceTo(en[i].getPosition()) <= weaponRange) {
          temp[j] = en[i];
          j++;
        }

      // now sort this array using simple swap sort
      swaps = 1;
      while (swaps != 0) {
        swaps = 0;
        for (i = 0; i < temp.length-1; i++) {
          if ( stsMon.getPosition().distanceTo(temp[i].getPosition()) >
stsMon.getPosition().distanceTo(temp[i+1].getPosition())) {
            t = temp[i];
            temp[i] = temp[i+1];
            temp[i+1] = t;
            swaps++;
          }
        }
      }
            return temp;
  }

  /*-------------------------------------------------------------------------*/

  private double maxActions(RLStateAction sa) {
            int i;
            double max = sa.values[0];

            for (i = 0; i < sa.values.length; i++) {
                      if (sa.values[i] > max) max = sa.values[i];
            }
            return max;
  }

  /*-------------------------------------------------------------------------*/

  private int getCurrentState() {
      // used for state determination
      int EiVR = 0;
      int WiVR = 0;
      int EiFR = 0;
      int WiFR = 0;
      int health = 0;
      int visualEFRatio = 0;
      int firingEFRatio = 0;
      double stateVals[] = new double[7];
      RLState tempState;

      // others
      double ratio;
      DetectedAgent en[], fr[];
      int frInRange, enInRange, i, index;
```

```
      double weaponRange;
      double avgDist;

      weaponRange = capMan.getCurrentWeapon().getMaxRange();
      en = infMan.getEnemies();     // get all the enemies in sight
      fr = infMan.getFriendlies();  // get all the friendlies in sight

      // are there enemies in Visual range
      if (infMan.getNoOfEnemies() > 0) EiVR = 1;

      // are there wounded in visual range
      if (EiVR == 1) {
        for (i = 0; i < infMan.getNoOfEnemies(); i++) {
          if (en[i].getHealth() < en[i].getMaxHealth()) WiVR++;
        }
        if (WiVR > 0) WiVR = 1;
      }

      // some preliminary stuff
      enInRange = 0; frInRange = 0;

      if (EiVR > 0) {
        for (i = 0; i < infMan.getNoOfEnemies(); i++)
          if (stsMon.getVectorTo(en[i].getPosition()).mag() <= weaponRange) enInRange++;
        for (i = 0; i < infMan.getNoOfFriendlies(); i++)
          if (stsMon.getVectorTo(fr[i].getPosition()).mag() <= weaponRange) frInRange++;
      }

      // are there enemies in firing range
      if (enInRange > 0) EiFR = 1;

      // are there wounded in firing range
      if (EiFR > 0) {
        for (i = 0; i < infMan.getNoOfEnemies(); i++)
          if ((stsMon.getVectorTo(en[i].getPosition()).mag() <= weaponRange) &&
(en[i].getHealth() < en[i].getMaxHealth())) WiFR++;
        if (WiFR > 0) WiFR = 1;
      }

      // work out the health
      if (stsMon.getPercentageHealth() < 0.5)
        health = 0;
      else
        health = 1;

      // work out the visualEFRatio the + 1 accounts for the current agent
      ratio = infMan.getNoOfEnemies() / (infMan.getNoOfFriendlies() + 1);
      if (ratio < 1)
        visualEFRatio = 0;  // more friendlies
      else
        if (ratio > 1)
          visualEFRatio = 2;  // more enemy
        else
          visualEFRatio = 1;  // equal numbers
```

```
      // work out the firingEFRatio the + 1 accounts for the current agent
      ratio = enInRange / (frInRange + 1);
      if (ratio < 1)
        firingEFRatio = 0;  // more friendlies
      else
        if (ratio > 1)
          firingEFRatio = 2;  // more enemy
        else
          firingEFRatio = 1;  // equal numbers


      //if (en.length > 0)
      //   System.out.println("I see "+en.length+" BAD GUY's   and "+fr.length+" GOOD
GUY's");
      //else
      //   System.out.println("All Clear    .except "+fr.length+" GOOD Guy's");

      //transfer the state variables to stateVals array
      stateVals[0] = EiVR;
      stateVals[1] = WiVR;
      stateVals[2] = EiFR;
      stateVals[3] = WiFR;
      stateVals[4] = health;
      stateVals[5] = visualEFRatio;
      stateVals[6] = firingEFRatio;

      // get the state index (if it exists)
      tempState = new RLState(stateVals);
      index = -1;
      for (i = 0; i < q.length; i++) {
            if (q[i].state.equals(tempState)) {
                  index = i;
                  break;
            }
      }
      if (index == -1) {
            // shouldn't happen... but just in case
        System.out.println("Major error **** state not found -- BasicRLAgent.java");
        System.exit(0);
      }
      return index;
  }

  /*-------------------------------------------------------------------------*/

  private void scan() {
    // if this agent has no sensor - then can't do anything
    if (capMan.getCurrentSensor() == null)
    {
      return;
    }
    // otherwise use the current scanner for default scan
    capMan.getCurrentSensor().scan();
  }
```

```
  /*---------------------------------------------------------------------*/

  // this method dumps the action-values to a file
  public void dumpStates() {
          int i,j;

          PrintStream out;

          try {
      out = new PrintStream(new FileOutputStream("c:\\crocadile\\zzactval.txt",true));
    } catch (FileNotFoundException e) {return;}
    out.print(stsMon.getID()); out.println();

    for (i = 0; i < q.length; i++) {
      out.print("state "+i+" ");
      for (j = 0; j < q[i].values.length; j++) {
        out.print(q[i].values[j]);
        out.print("  ");
      }
      out.println();
    }
    out.println("-----------------------------------------------------");
    out.close();
  }
}
```

## A.12 RLStateAction Object Code Listing

```
/*
   This class represents a Reinforcement learning StateAction class.  It contains an object
   representing a Reinforcement  Learning State, and two arrays.... one representing the
   list of all possible actions available in that state .... and another array representing
   the learned action-values for the corresponding actions.

   Author: Paul Darbyshire
   version 2
*/

package ares;

import java.io.*;

public class RLStateAction implements Serializable {

    public RLState state;      // represents a 'State' an agent can be in
    public String actions[];   // represents the possible actions in this state
    double values[];      // represents the action-values for this state.

    public RLStateAction( RLState s, String a[]) {
        int i;

        state = s;
        actions = a;
        values = new double[actions.length];
        for (i = 0; i < values.length; i++) values[i] = 0;
    }

    // added in for a quick check
    public RLStateAction( RLState s, String a[], double b0, double b1, double b2, double b3,
double b4, double b5) {
        int i;

        state = s;
        actions = a;
        values = new double[actions.length];
        values[0] = b0;
        values[1] = b1;
        values[2] = b2;
        values[3] = b3;
        values[4] = b4;
        values[5] = b5;
    }

}
```

## A.13 RLState Object Code Listing

```
/*
   This class represents a Reinforcement learning state.  It only contains attributes
   representing the fundamental characteristics comprising the state and a simple method
   to compare this state to another state passed as a parameter.

   Author: Paul Darbyshire
   Version 2
*/

package ares;

import java.io.*;

public class RLState implements Serializable {

     /* State variables */
     public int EiVR; // Enemy in Visual Range  two states       0 - No   1 - Yes
     public int WiVR; // Wounded  in Visual Range  two states    0 - No   1 - Yes
     public int EiFR; // Enemy in Firing Range  two states       0 - No   1 - Yes
     public int WiFR; // Wounded  in Firing Range  two states    0 - No   1 - Yes

    /* this is the quantized health state... crrently has two states
       0  (bad  - percentage health <50%)
       1  (good - percentage health >= 50%)
    */
    public int health;

    /* quantized value for quantity  e/f  in visual range
       0  (e/f ratio in interval [0,1)           ... more friendlies)
       1  (e/f ratio    is 1                   ... equal amounts)
       2  (e/f ratio is in interval (1, infinity) ... more enemy )
    */
    public int visualEFRatio;

    /* quantized value for quantity  e/f  in firing range
       0  (e/f ratio in interval [0,1)  )
       1  (e/f ratio    is 1  )
       2  (e/f ratio is in interval (1, infinity)  )
    */
    public int firingEFRatio;

    public RLState(double values[]) {
          if (values.length != 7) {
             System.out.println("Invalid Values array passed to RLState constructor");
             System.exit(0);
          }
          EiVR = (int) values[0];
          WiVR = (int) values[1];
          EiFR = (int) values[2];
          WiFR = (int) values[3];
        health = (int) values[4];
```

```
        visualEFRatio = (int) values[5];
        firingEFRatio = (int) values[6];
    }

    boolean equals(RLState s) {
            if (EiVR != s.EiVR) return false;
            if (WiVR != s.WiVR) return false;
            if (EiFR != s.EiFR) return false;
            if (WiFR != s.WiFR) return false;
        if (health != s.health) return false;
        if (visualEFRatio != s.visualEFRatio) return false;
        if (firingEFRatio != s.firingEFRatio) return false;
        return true;
    }
}
```

## A.14 RLUtil Object Code Listing

```
package ares;

/*  The purpose of this class is to open the StateActionMap text file to read
    the different states and actions available, so as to build the states for
    access by the Reinforcement Learning bases agents

   Author: Paul Darbyshire
   version 2

*/

import java.io.*;
import java.util.Vector;

class RLUtil {
   /*
   public static void main(String args[]) throws IOException {
            int tt, i,j;
            String strtok;
            double numtoc;

            int numStateValues;
            double stateValues[];
            Vector actions = new Vector();
            Vector stateActions = new Vector();
            String stateActionsStr[];
            Vector stateActionObjs = new Vector();

            RLState s;
            RLStateAction sa;
            RLStateAction q[];

            Reader r = new BufferedReader(new InputStreamReader(new
FileInputStream("c:\\crocadile\\StateActionMap.txt")));
      StreamTokenizer st = new StreamTokenizer(r);

      //DataInputStream in = new DataInputStream(new
FileInputStream("StateActionMap.txt"));

      st.slashSlashComments(true);
      st.slashStarComments(true);

      // get values: and the corresponding number
      tt = st.nextToken();
      if (tt != StreamTokenizer.TT_WORD) error(0,st);
      if (!st.sval.equals("values")) error(0,st);
      tt = st.nextToken();
      if (((char) tt) != ':') error(1,st);
      tt = st.nextToken();
      if (tt != StreamTokenizer.TT_NUMBER) error(2,st);
      numStateValues = (int) st.nval;
```

```
// get actions: and the corresponding list of actions
tt = st.nextToken();
if (tt != StreamTokenizer.TT_WORD) error(3,st);
if (!st.sval.equals("actions")) error(3,st);
tt = st.nextToken();
if (((char) tt) != ':') error(1,st);
// now get the list of actions
do {
   tt = st.nextToken();
   if (tt != StreamTokenizer.TT_WORD) error(4,st);
   if (!st.sval.equals("states")) actions.addElement(st.sval);
} while (!st.sval.equals("states"));

// get the : after the states
tt = st.nextToken();
if (((char) tt) != ':') error(1,st);

//now loop to get all the states and associated actions
tt = st.nextToken();
while (tt != StreamTokenizer.TT_EOF) {
   if (((char) tt) != '(') error(5,st);

   //now get all the state values)
   stateValues = new double[numStateValues];
   for (i = 0; i < numStateValues; i++) {
      tt = st.nextToken();
      if (tt != StreamTokenizer.TT_NUMBER) error(2,st);
     stateValues[i] = st.nval;
   }
   tt = st.nextToken();
   if (((char) tt) != ')') error(6,st);

   // create the RLState object
   s = new RLState(stateValues);

   // now get all the actions for this state
   stateActions.removeAllElements();
   tt = st.nextToken();
   while (tt == StreamTokenizer.TT_WORD) {
     if (!validAction(st.sval, actions)) error(7,st);
     stateActions.addElement(st.sval);
      tt = st.nextToken();
   }

   //create an action String array from the vector
   stateActionsStr = new String[stateActions.size()];
   for (i = 0; i < stateActions.size(); i++)
      stateActionsStr[i] = (String) stateActions.elementAt(i);

   // create the RLStateAction object
   sa = new RLStateAction(s, stateActionsStr);
   stateActionObjs.addElement(sa);
 }
```

```
    // now finally construct the stateAction Array
    q = new RLStateAction[stateActionObjs.size()];
    for (i = 0; i < stateActionObjs.size(); i++)
      q[i] = (RLStateAction) stateActionObjs.elementAt(i);

     // now print it all out
     for (i = 0; i < q.length; i++) {
       System.out.print("State values: "+q[i].state.EiVR+" "+q[i].state.WiVR+"
"+q[i].state.EiFR+" "+q[i].state.WiFR+" "+q[i].state.health+" "+q[i].state.visualEFRatio+"
"+q[i].state.firingEFRatio+" ");
       System.out.print("actions: ");
       for (j = 0; j < q[i].actions.length; j++) System.out.print(q[i].actions[j]+" ");
       System.out.println();
     }
     System.out.println("\n.....\n");
     System.out.println("Total number of states recorded is: "+q.length);
     System.out.println("The number of state values is "+numStateValues);
     System.out.print("The possible actions are: ");
     for (i = 0; i < actions.size(); i++) {
           strtok = (String) actions.elementAt(i);
           System.out.print(strtok+"  ");
     }
     System.out.println();


     r.close();

     // at last.... return the State-Action array q
     //return q;
   }
   */
   public static RLStateAction[] buildQ() throws IOException {
           int tt, i,j;
           String strtok;
           double numtoc;

           int numStateValues;
           double stateValues[];
           Vector actions = new Vector();
           Vector stateActions = new Vector();
           String stateActionsStr[];
           Vector stateActionObjs = new Vector();

           RLState s;
           RLStateAction sa;
           RLStateAction q[];

           Reader r = new BufferedReader(new InputStreamReader(new
FileInputStream("c:\\crocadile\\StateActionMap.txt")));
     StreamTokenizer st = new StreamTokenizer(r);

     //DataInputStream in = new DataInputStream(new
FileInputStream("StateActionMap.txt"));

     st.slashSlashComments(true);
```

```
st.slashStarComments(true);

// get values: and the corresponding number
tt = st.nextToken();
if (tt != StreamTokenizer.TT_WORD) error(0,st);
if (!st.sval.equals("values")) error(0,st);
tt = st.nextToken();
if (((char) tt) != ':') error(1,st);
tt = st.nextToken();
if (tt != StreamTokenizer.TT_NUMBER) error(2,st);
numStateValues = (int) st.nval;

// get actions: and the corresponding list of actions
tt = st.nextToken();
if (tt != StreamTokenizer.TT_WORD) error(3,st);
if (!st.sval.equals("actions")) error(3,st);
tt = st.nextToken();
if (((char) tt) != ':') error(1,st);
// now get the list of actions
do {
   tt = st.nextToken();
   if (tt != StreamTokenizer.TT_WORD) error(4,st);
   if (!st.sval.equals("states")) actions.addElement(st.sval);
} while (!st.sval.equals("states"));

// get the : after the states
tt = st.nextToken();
if (((char) tt) != ':') error(1,st);

//now loop to get all the states and associated actions
tt = st.nextToken();
while (tt != StreamTokenizer.TT_EOF) {
   if (((char) tt) != '(') error(5,st);

   //now get all the state values)
   stateValues = new double[numStateValues];
   for (i = 0; i < numStateValues; i++) {
      tt = st.nextToken();
      if (tt != StreamTokenizer.TT_NUMBER) error(2,st);
     stateValues[i] = st.nval;
   }
   tt = st.nextToken();
   if (((char) tt) != ')') error(6,st);

   // create the RLState object
   s = new RLState(stateValues);

   // now get all the actions for this state
   stateActions.removeAllElements();
   tt = st.nextToken();
   while (tt == StreamTokenizer.TT_WORD) {
     if (!validAction(st.sval, actions)) error(7,st);
     stateActions.addElement(st.sval);
      tt = st.nextToken();
   }
```

```
        //create an action String array from the vector
        stateActionsStr = new String[stateActions.size()];
        for (i = 0; i < stateActions.size(); i++)
          stateActionsStr[i] = (String) stateActions.elementAt(i);

        // create the RLStateAction object
        sa = new RLStateAction(s, stateActionsStr);
        stateActionObjs.addElement(sa);
       }

     // now finally construct the stateAction Array
     q = new RLStateAction[stateActionObjs.size()];
     for (i = 0; i < stateActionObjs.size(); i++)
        q[i] = (RLStateAction) stateActionObjs.elementAt(i);

      // now print it all out
      /*for (i = 0; i < q.length; i++) {
        System.out.print("State values: "+q[i].state.EiVR+" "+q[i].state.WiVR+"
"+q[i].state.EiFR+" "+q[i].state.WiFR+" "+q[i].state.health+" "+q[i].state.visualEFRatio+"
"+q[i].state.firingEFRatio+" ");
        System.out.print("actions: ");
        for (j = 0; j < q[i].actions.length; j++) System.out.print(q[i].actions[j]+" ");
        System.out.println();
      }
      System.out.println("\n.....\n");
      System.out.println("Total number of states recorded is: "+q.length);
      System.out.println("The number of state values is "+numStateValues);
      System.out.print("The possible actions are: ");
      for (i = 0; i < actions.size(); i++) {
            strtok = (String) actions.elementAt(i);
            System.out.print(strtok+"  ");
      }
      System.out.println();
      */

     r.close();

     // at last.... return the State-Action array q
     return q;
   }

   static boolean validAction(String strtok, Vector actions) {
            int i;

            for (i = 0; i < actions.size(); i++) {
              if (strtok.equals((String) actions.elementAt(i))) return true;
            }
            return false;
   }

   static void error(int errnum, StreamTokenizer st) {
            System.out.print("error in line "+st.lineno()+"...");
     if (errnum == 0)       System.out.println("Expected 'values'");
     else if (errnum == 1)  System.out.println("Expected ':'");
```

```
    else if (errnum == 2)  System.out.println("Expected a numeric value");
    else if (errnum == 3)  System.out.println("Expected 'actions'");
    else if (errnum == 4)  System.out.println("Expected an action token");
    else if (errnum == 5)  System.out.println("Expected '('");
    else if (errnum == 6)  System.out.println("Expected ')'");
    else if (errnum == 7)  System.out.println("Not a valid action");

    System.exit(0);
  }

}
```

# APPENDIX B

# SIMULATION ELEMENTS

The following pages in this Appendix contain elements of the simulation not included in the Simulation Code Appendix. This includes the State Action Map in B.1 which is used by the simulation engine upon entry to initialize the reinforcement learning agents. Sections B.2 – B.5 contain excerpts of the output files generated by the simulation. The complete files are not listed here as the output volume would be too large to reasonably publish. However, the full output files can be supplied on request in electronic form.

## B.1 State Action Map

/* Current list of possible actions Ver 2 */

values: 7

actions: nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW

/* states are given by tupples of the form
(EiVR, WiVR, EiFR, WiFR, heath, VisualEFRatio, firingEFRatio)

EiVR Enemy in Visual Range two states 0 - No 1 - Yes

WiVR Wounded in Visual Range two states 0 - No 1 - Yes

EiFR Enemy in Firing Range two states 0 - No 1 - Yes

WiFR Wounded in Firing Range two states 0 - No 1 - Yes

health this is the quantized health state... currently has two states
0 (bad - percentage health <50%)
1 (good - percentage health >= 50%)

visualEFRatio quantized value for quantity e/f in visual range
0 (e/f ratio in interval [0,1) ... more friendlies)
1 (e/f ratio is 1 ... equal amounts)
2 (e/f ratio is in interval (1, infinity) ... more enemy )

firingEFRatio quantized value for quantity e/f in firing range
0 (e/f ratio in interval [0,1) ... more friendlies)
1 (e/f ratio is 1 ... equal amounts)
2 (e/f ratio is in interval (1, infinity) ... more enemy )

*/

states:

(0 0 0 0 0 0 0) nothing explore
(0 0 0 0 1 0 0) nothing explore

(1 0 0 0 0 0 0) nothing explore movaway movto
(1 0 0 0 0 1 0) nothing explore movaway movto
(1 0 0 0 0 2 0) nothing explore movaway movto
(1 0 0 0 1 0 0) nothing explore movaway movto
(1 0 0 0 1 1 0) nothing explore movaway movto
(1 0 0 0 1 2 0) nothing explore movaway movto

(1 0 1 0 0 0 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 0 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 0 2) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 1 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 1 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 1 2) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 2 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 2 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 0 2 2) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 1 0 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 1 0 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0 1 0 2) nothing explore movaway movto fire movtoF movawayF

(1 0 1 0  1 1 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  1 1 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  1 1 2) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  1 2 0) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  1 2 1) nothing explore movaway movto fire movtoF movawayF
(1 0 1 0  1 2 2) nothing explore movaway movto fire movtoF movawayF

(1 1 0 0  0 0 0) nothing explore movaway movto movtoW
(1 1 0 0  0 1 0) nothing explore movaway movto movtoW
(1 1 0 0  0 2 0) nothing explore movaway movto movtoW
(1 1 0 0  1 0 0) nothing explore movaway movto movtoW
(1 1 0 0  1 1 0) nothing explore movaway movto movtoW
(1 1 0 0  1 2 0) nothing explore movaway movto movtoW

(1 1 1 0  0 0 0) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 0 1) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 0 2) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 1 0) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 1 1) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 1 2) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 2 0) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 2 1) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  0 2 2) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 0 0) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 0 1) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 0 2) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 1 0) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 1 1) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 1 2) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 2 0) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 2 1) nothing explore movaway movto fire movtoW movtoF movawayF
(1 1 1 0  1 2 2) nothing explore movaway movto fire movtoW movtoF movawayF

(1 1 1 1  0 0 0) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 0 1) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 0 2) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 1 0) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 1 1) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 1 2) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 2 0) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 2 1) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  0 2 2) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 0 0) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 0 1) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 0 2) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 1 0) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 1 1) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 1 2) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 2 0) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 2 1) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW
(1 1 1 1  1 2 2) nothing explore movaway movto fire movtoW fireW movtoF movtoFW movawayF movawayFW

## B.2 CRT output file

This output file records the health state of each team at the end of each iteration of the main simulation loop. The format of the crt file is viewed in the extract below. The first column represents the simulation time, the following columns represent, the team identification number, the total remaining health points of the team, the remaining number of agents in the team, the opponent team identification, the remaining health points of the opponent team and the number of agents left alive in the opponent team.

The following is an extract of the 141crt.csv file generated during simulation run 141 of one of the four simulation trials

```
                    :
                    :
97    0    1500    15    1    1500    15
98    0    1485    15    1    1485    15
99    0    1485    15    1    1485    15
100   0    1485    15    1    1485    15
101   0    1485    15    1    1485    15
102   0    1470    15    1    1485    15
103   0    1455    15    1    1440    15
104   0    1440    15    1    1440    15
105   0    1440    15    1    1440    15
106   0    1440    15    1    1440    15
107   0    1425    15    1    1425    15
108   0    1410    15    1    1395    15
109   0    1410    15    1    1395    15
110   0    1410    15    1    1395    15
111   0    1400    14    1    1395    15
112   0    1370    14    1    1395    15
113   0    1370    14    1    1395    15
114   0    1355    14    1    1365    15
115   0    1355    14    1    1350    15
116   0    1325    14    1    1335    15
117   0    1295    14    1    1320    15
118   0    1265    14    1    1290    15
119   0    1250    14    1    1290    15
120   0    1250    14    1    1260    15
121   0    1210    13    1    1245    15
122   0    1180    13    1    1235    14
123   0    1150    13    1    1220    14
124   0    1120    13    1    1190    14
125   0    1120    13    1    1175    14
126   0    1090    13    1    1160    14
```

| | | | | | | |
|---|---|---|---|---|---|---|
| 127 | 0 | 1075 | 13 | 1 | 1090 | 13 |
| 128 | 0 | 1045 | 13 | 1 | 1075 | 13 |
| 129 | 0 | 1030 | 13 | 1 | 1075 | 13 |
| 130 | 0 | 985 | 13 | 1 | 1075 | 13 |
| 131 | 0 | 970 | 13 | 1 | 1045 | 13 |
| 132 | 0 | 955 | 13 | 1 | 1045 | 13 |
| 133 | 0 | 930 | 12 | 1 | 990 | 12 |
| 134 | 0 | 915 | 12 | 1 | 990 | 12 |
| 135 | 0 | 900 | 12 | 1 | 990 | 12 |
| 136 | 0 | 885 | 12 | 1 | 990 | 12 |
| 137 | 0 | 870 | 12 | 1 | 930 | 12 |
| 138 | 0 | 845 | 11 | 1 | 930 | 12 |
| 139 | 0 | 805 | 10 | 1 | 930 | 12 |
| 140 | 0 | 790 | 10 | 1 | 930 | 12 |
| 141 | 0 | 775 | 10 | 1 | 930 | 12 |
| 142 | 0 | 775 | 10 | 1 | 930 | 12 |
| 143 | 0 | 735 | 9 | 1 | 920 | 11 |
| 144 | 0 | 720 | 9 | 1 | 905 | 11 |
| 145 | 0 | 705 | 9 | 1 | 905 | 11 |
| 146 | 0 | 705 | 9 | 1 | 905 | 11 |
| 147 | 0 | 675 | 9 | 1 | 905 | 11 |
| 148 | 0 | 675 | 9 | 1 | 905 | 11 |
| 149 | 0 | 675 | 9 | 1 | 890 | 11 |
| 150 | 0 | 675 | 9 | 1 | 890 | 11 |
| 151 | 0 | 675 | 9 | 1 | 875 | 11 |
| 152 | 0 | 660 | 9 | 1 | 875 | 11 |
| 153 | 0 | 660 | 9 | 1 | 860 | 11 |
| 154 | 0 | 660 | 9 | 1 | 850 | 10 |
| 155 | 0 | 660 | 9 | 1 | 850 | 10 |
| 156 | 0 | 650 | 8 | 1 | 850 | 10 |
| 157 | 0 | 650 | 8 | 1 | 835 | 10 |
| 158 | 0 | 620 | 8 | 1 | 835 | 10 |
| 159 | 0 | 620 | 8 | 1 | 835 | 10 |
| 160 | 0 | 620 | 8 | 1 | 835 | 10 |
| 161 | 0 | 620 | 8 | 1 | 825 | 9 |
| 162 | 0 | 590 | 8 | 1 | 825 | 9 |
| 163 | 0 | 590 | 8 | 1 | 825 | 9 |
| 164 | 0 | 560 | 8 | 1 | 810 | 9 |
| 165 | 0 | 560 | 8 | 1 | 810 | 9 |
| 166 | 0 | 560 | 8 | 1 | 810 | 9 |
| 167 | 0 | 545 | 8 | 1 | 810 | 9 |
| | | | ⋮ | | | |
| | | | ⋮ | | | |

## B.3 CRE Output file

This output file captures relevant information about a 'Hit', damage inflicted to another agent as a result of an agent action, gathered during the processing of the *actions*[ ] array after all the agents have completed their thinking cycles. The format of the cre file is viewed in the extract below. The first column shown is the agent identification of the agent being 'Hit', followed by its agent team identification, the identification of the attacking agent (needed to allocate rewards), the attackers team identification, the cause of the damage (currently only 1 is valid), the amount of damage in health points (usually 15), the simulation time, followed by the (x,z,y) coordinates of the agent being 'Hit'. The (x,z,y) coordinates are the coordinates of the agent in the simulated world, with the z coordinate always 2.5 in the current simulation.

The following is an extract of the 141cre.csv file generated during simulation run 141 of one of the four simulation trials

:
:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 11 | 0 | 1 | 15 | 97 | 257.8122977 | 2.5 | 232.0120181 |
| 27 | 1 | 4 | 0 | 1 | 15 | 97 | 336.7020277 | 2.5 | 333.1477133 |
| 4 | 0 | 28 | 1 | 1 | 15 | 101 | 267.2886271 | 2.5 | 244.9038441 |
| 4 | 0 | 27 | 1 | 1 | 15 | 102 | 269.6569372 | 2.5 | 248.1273682 |
| 27 | 1 | 15 | 1 | 1 | 15 | 102 | 334.5380405 | 2.5 | 329.7386828 |
| 22 | 1 | 4 | 0 | 1 | 15 | 102 | 330.3138412 | 2.5 | 361.5746663 |
| 26 | 1 | 29 | 1 | 1 | 15 | 102 | 323.7876139 | 2.5 | 342.4282157 |
| 4 | 0 | 26 | 1 | 1 | 15 | 103 | 272.0249382 | 2.5 | 251.3511192 |
| 4 | 0 | 27 | 1 | 1 | 15 | 106 | 279.1381997 | 2.5 | 261.0155633 |
| 21 | 1 | 19 | 1 | 1 | 15 | 106 | 352.3609714 | 2.5 | 334.0668851 |
| 4 | 0 | 28 | 1 | 1 | 15 | 107 | 281.5116504 | 2.5 | 264.2353042 |
| 27 | 1 | 15 | 1 | 1 | 15 | 107 | 332.3147524 | 2.5 | 326.3763337 |
| 18 | 1 | 4 | 0 | 1 | 15 | 107 | 349.0797507 | 2.5 | 359.7840282 |
| 4 | 0 | 29 | 1 | -1 | 15 | 110 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 29 | 1 | 1 | 15 | 110 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 28 | 1 | 1 | 15 | 111 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 27 | 1 | 1 | 15 | 111 | 288.6389437 | 2.5 | 273.8894065 |
| 8 | 0 | 7 | 0 | 1 | 15 | 111 | 250.0914752 | 2.5 | 264.4895332 |
| 11 | 0 | 6 | 0 | 1 | 15 | 111 | 282.1760662 | 2.5 | 258.3343467 |
| 4 | 0 | 26 | 1 | 1 | 15 | 112 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 25 | 1 | 1 | 15 | 112 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 22 | 1 | 1 | 15 | 112 | 288.6389437 | 2.5 | 273.8894065 |
| 2 | 0 | 1 | 0 | 1 | 15 | 113 | 248.9093303 | 2.5 | 236.2298816 |
| 4 | 0 | 21 | 1 | 1 | 15 | 113 | 288.6389437 | 2.5 | 273.8894065 |
| 27 | 1 | 15 | 1 | 1 | 15 | 113 | 328.8708894 | 2.5 | 321.4239534 |
| 18 | 1 | 10 | 0 | 1 | 15 | 113 | 342.7881348 | 2.5 | 352.0864725 |
| 24 | 1 | 19 | 1 | 1 | 15 | 114 | 368.4350159 | 2.5 | 348.1783557 |
| 4 | 0 | 29 | 1 | 1 | 15 | 115 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 2 | 0 | 1 | 15 | 115 | 288.6389437 | 2.5 | 273.8894065 |
| 10 | 0 | 2 | 0 | 1 | 15 | 115 | 267.1379962 | 2.5 | 257.4199658 |
| 11 | 0 | 27 | 1 | 1 | 15 | 115 | 291.2499244 | 2.5 | 271.5125465 |
| 22 | 1 | 23 | 1 | 1 | 15 | 115 | 321.0606312 | 2.5 | 344.0650449 |
| 6 | 0 | 5 | 0 | 1 | 15 | 116 | 266.8128882 | 2.5 | 225.3810325 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 6 | 0 | 1 | 15 | 116 | 288.6389437 | 2.5 | 273.8894065 |
| 11 | 0 | 18 | 1 | 1 | 15 | 116 | 293.5270018 | 2.5 | 274.8011508 |
| 27 | 1 | 12 | 0 | 1 | 15 | 116 | 333.3360234 | 2.5 | 325.4744177 |
| 4 | 0 | 17 | 1 | 1 | 15 | 117 | 288.6389437 | 2.5 | 273.8894065 |
| 11 | 0 | 28 | 1 | 1 | 15 | 117 | 295.811105 | 2.5 | 278.0848793 |
| 11 | 0 | 25 | 1 | 1 | 15 | 117 | 295.811105 | 2.5 | 278.0848793 |
| 22 | 1 | 10 | 0 | 1 | 15 | 117 | 319.5239891 | 2.5 | 340.3720224 |
| 26 | 1 | 10 | 0 | 1 | 15 | 117 | 319.1869821 | 2.5 | 339.2708547 |
| 2 | 0 | 1 | 0 | 1 | 15 | 118 | 261.8377247 | 2.5 | 251.489515 |
| 27 | 1 | 15 | 1 | 1 | 15 | 119 | 329.6472117 | 2.5 | 320.7424357 |
| 18 | 1 | 28 | 1 | 1 | 15 | 119 | 341.0687102 | 2.5 | 348.3890885 |
| 10 | 0 | 21 | 1 | 1 | 15 | 120 | 284.5214192 | 2.5 | 266.1379045 |
| 11 | 0 | 29 | 1 | 1 | 15 | 120 | 302.7177041 | 2.5 | 287.8980469 |
| 11 | 0 | 27 | 1 | -1 | 15 | 120 | 302.7177041 | 2.5 | 287.8980469 |
| 11 | 0 | 27 | 1 | 1 | 15 | 120 | 302.7177041 | 2.5 | 287.8980469 |
| 24 | 1 | 19 | 1 | 1 | 15 | 120 | 365.7042407 | 2.5 | 345.2460636 |
| 8 | 0 | 3 | 0 | 1 | 15 | 121 | 279.7251303 | 2.5 | 291.3566305 |
| 10 | 0 | 16 | 1 | 1 | 15 | 121 | 287.3115455 | 2.5 | 269.0041207 |
| 11 | 0 | 18 | 1 | 1 | 15 | 121 | 302.7177041 | 2.5 | 287.8980469 |
| 27 | 1 | 13 | 0 | -1 | 15 | 121 | 327.1761439 | 2.5 | 317.5969908 |
| 27 | 1 | 13 | 0 | 1 | 15 | 121 | 327.1761439 | 2.5 | 317.5969908 |
| 4 | 0 | 5 | 0 | 1 | 15 | 122 | 288.6389437 | 2.5 | 273.8894065 |
| 4 | 0 | 20 | 1 | 1 | 15 | 122 | 288.6389437 | 2.5 | 273.8894065 |
| 10 | 0 | 5 | 0 | 1 | 15 | 122 | 290.0753445 | 2.5 | 271.8957319 |
| 10 | 0 | 20 | 1 | 1 | 15 | 122 | 290.0753445 | 2.5 | 271.8957319 |
| 11 | 0 | 25 | 1 | 1 | 15 | 122 | 302.7177041 | 2.5 | 287.8980469 |
| 11 | 0 | 22 | 1 | 1 | 15 | 122 | 302.7177041 | 2.5 | 287.8980469 |
| 27 | 1 | 8 | 0 | 1 | 15 | 122 | 327.1761439 | 2.5 | 317.5969908 |
| 25 | 1 | 29 | 1 | 1 | 15 | 122 | 347.6054966 | 2.5 | 379.605354 |
| 2 | 0 | 1 | 0 | 1 | 15 | 123 | 274.8168042 | 2.5 | 266.7060555 |
| 7 | 0 | 13 | 0 | 1 | 15 | 123 | 258.0135229 | 2.5 | 277.0429345 |
| 11 | 0 | 23 | 1 | 1 | 15 | 123 | 302.7177041 | 2.5 | 287.8980469 |
| 26 | 1 | 7 | 0 | 1 | 15 | 123 | 320.5646172 | 2.5 | 340.8419662 |
| 26 | 1 | 6 | 0 | 1 | 15 | 123 | 320.5646172 | 2.5 | 340.8419662 |
| 24 | 1 | 20 | 1 | 1 | 15 | 124 | 362.7883662 | 2.5 | 342.5078752 |
| 2 | 0 | 3 | 0 | 1 | 15 | 125 | 280.0196289 | 2.5 | 272.7831119 |
| 8 | 0 | 18 | 1 | 1 | 15 | 125 | 291.5033248 | 2.5 | 302.1859119 |
| 26 | 1 | 12 | 0 | 1 | 15 | 125 | 318.0449766 | 2.5 | 337.7352942 |
| 8 | 0 | 22 | 1 | 1 | 15 | 126 | 294.4436862 | 2.5 | 304.8977886 |
| 22 | 1 | 28 | 1 | 1 | 15 | 126 | 309.9719871 | 2.5 | 325.4243995 |
| 22 | 1 | 8 | 0 | 1 | 15 | 126 | 309.9719871 | 2.5 | 325.4243995 |
| 27 | 1 | 9 | 0 | 1 | 15 | 126 | 327.1761439 | 2.5 | 317.5969908 |
| 18 | 1 | 15 | 1 | 1 | 15 | 126 | 331.62029 | 2.5 | 338.1761454 |
| 26 | 1 | 25 | 1 | 1 | 15 | 126 | 316.7851562 | 2.5 | 336.1819582 |
| 26 | 1 | 2 | 0 | -1 | 15 | 126 | 316.7851562 | 2.5 | 336.1819582 |
| 26 | 1 | 2 | 0 | 1 | 15 | 126 | 316.7851562 | 2.5 | 336.1819582 |
| 4 | 0 | 5 | 0 | 1 | 15 | 127 | 288.6389437 | 2.5 | 273.8894065 |
| 7 | 0 | 14 | 0 | 1 | 15 | 127 | 270.1092368 | 2.5 | 287.5163928 |

:
:

## B.4 ACTVAL file output

This output file records the state of each agents *Q*[ ] array at the end of an entire simulation run, that is the values inside a reinforcement learning agents state-action matrix. The format of the actval file is viewed in the extract below. The first line contains the identification number of the agent, every subsequent line contains the text 'state', followed by the state number (valid values are 0 -67) allowing for 68 states. The remaining columns are a dump of the *values*[ ] array for that agent (see *Chapter 4 Figure 4.13*) where each value corresponds to allowed action for that state. Each state can have a different number of allowed actions so the remaining column numbers are not consistent. This file should be read in conjunction with the *StateActionMap* in *Section B.1* of this appendix.

The following is an extract of the 141actval.txt file generated during simulation run 141 of one of the four simulation trials

```
15
state 0 0.0  0.0
state 1 0.0  0.0
state 2 0.0  0.0  0.0  0.0
state 3 0.0  0.0  0.0  0.0
state 4 0.0  0.0  0.0  0.0
state 5 0.0  0.0  0.0  0.0
state 6 0.0  0.0  0.0  0.0
state 7 0.0  0.0  0.0  0.0
state 8 0.0  0.0  -0.049495000000000004  0.0  0.0  0.0  0.0
state 9 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 10 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 11 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 12 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 13 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 14 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 15 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 16 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 17 0.0  -0.095  0.0  0.0  0.0  0.09700132413255001  0.0
state 18 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 19 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 20 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 21 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 22 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 23 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 24 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 25 0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 26 0.0  0.0  0.0  0.0  0.0
state 27 0.0  0.0  0.0  0.0  0.0
state 28 0.0  0.0  0.0  0.0  0.0
state 29 0.0  0.0  0.0  0.0  0.0
state 30 0.0  0.0  0.0  0.0  0.0
state 31 0.0  0.0  0.0  0.0  0.0
state 32 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 33 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 34 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 35 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 36 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
state 37 0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

state 38 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 39 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 40 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 41 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 42 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 43 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 44 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 45 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 46 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 47 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 48 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 49 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 50 -0.049577767188510063 -0.0486462295 -0.04779561065614972 -0.095
-0.001974843049999994 -0.05 -0.07933397127724105 -0.08978138484208491 0.11199119122550001
-0.0320077645 0.0
state 51 0.0 0.0 -0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 52 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 53 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0678577811995 0.0 0.0 0.0
state 54 0.0 0.0 0.0 0.0 -0.05405000000000001 0.0 0.0 0.0554589604076 0.09231396040759998
-0.05 0.0
state 55 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 56 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 57 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 58 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 59 0.0 0.0 -0.0495 0.0 0.0 0.0 0.25506234867287664 0.10437916615526574 0.05
0.3404278989629099 0.05
state 60 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 61 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 62 0.0 0.0 0.0 0.0 0.09769064426905 0.0 0.0 0.0 0.0 0.35006286637801376 -0.05
state 63 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 64 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 65 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 66 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 67 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
---
16
state 0 0.0 0.0
state 1 0.0 0.0
state 2 0.0 0.0 0.0 0.0
state 3 0.0 0.0 0.0 0.0
state 4 0.0 0.0 0.0 0.0
state 5 0.0 0.0 0.0 0.0
state 6 0.0 0.0 0.0 0.0
state 7 0.0 0.0 0.0 0.0
state 8 0.0 0.0 -0.049495000000000004 0.0 0.0 0.0 0.0
state 9 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 10 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 11 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 12 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 13 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 14 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 15 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 16 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 17 0.0 -0.095 0.0 0.0 0.0 0.096552154554 0.0
state 18 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 19 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 20 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 21 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 22 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 23 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 24 0.0 0.0 0.0 0.0 0.0 0.0 0.0

state 25 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 26 0.0 0.0 0.0 0.0 0.0
state 27 0.0 0.0 0.0 0.0 0.0
state 28 0.0 0.0 0.0 0.0 0.0
state 29 0.0 0.0 0.0 0.0 0.0
state 30 0.0 0.0 0.0 0.0 0.0
state 31 0.0 0.0 0.0 0.0 0.0
state 32 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 33 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 34 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 35 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 36 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 37 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 38 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 39 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 40 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 41 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 42 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 43 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 44 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 45 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 46 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 47 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 48 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 49 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 50 -0.049582464952981 -0.04927655655 -0.04817726441538865 -0.095
-0.001974843049999994 -0.05 -0.06947302346371048 -0.10464889047619358 0.13491035722801903
-0.068250189 0.0
state 51 0.0 0.0 -0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 52 0.0 0.0 -0.05 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 53 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.02604251502954999 0.0 0.0 0.0
state 54 0.0 0.0 0.0 0.0 -0.05405000000000001 0.0 0.0 0.055446984193650004
0.09230198419364999 -0.05 0.0
state 55 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 56 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 57 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 58 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 59 0.0 0.0 -0.0495 0.0 0.0 0.0 0.31208309479307494 0.10456047067477359 0.05
0.36987867211979425 0.05
state 60 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 61 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 62 0.0 0.0 0.0 0.0 0.097690598855 0.0 0.0 0.0 0.0 0.3504014566505321 -0.05
state 63 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 64 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 65 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 66 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
state 67 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
---
17
state 0 0.0 0.0
state 1 0.0 0.0
state 2 0.0 0.0 0.0 0.0
state 3 0.0 0.0 0.0 0.0

:
:

## B.5 SCREEN file output

This output file shows information displayed to the screen such a rewards found and messages sent to other agents during the simulation run. The format of the ***screen*** file is viewed in the extract below. This is used mainly as a counter check to ensure the simulation is running correctly and can be compared to the cre and crt files to ensure specific events coincide at the correct simulation times. An entry is sent to the screen every time an agent receives a reward, containing the simulation time it received the reward, the agent identification number, the amount of the reward and the simulation time the reward was actually generated. Additionally, every time an agent sends a message to another agent, a slightly indented screen message is generated consisting of the simulation time, the agent identification of the agent to whom the message is being sent, and the other parameters which enable the receiving agent to update its state-action matrix, such as the state and action identifications, the reward, the new state identification and the agent that sent the message.

The following is an extract of the 141screen.txt file generated during simulation run 141 of one of the four simulation trials

```
End time = 1000.0
98: 27: found reward = -0.5  from time 92.0
   99: 15 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 16 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 17 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 18 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 19 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 20 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 21 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 22 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 23 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 24 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 25 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 26 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 27 - State 17  Action 1  Reward -0.5  newState 50  from 27
   99: 28 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
   99: 29 - State 17  Action 1  Reward -0.5  newState 50  from 27   17 1 -0.5 50 27
102: 28: found reward = 0.5  from time 97.0
103: 15: found reward = 0.5  from time 100.0
   103: 15 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 16 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 17 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 18 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 19 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 20 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 21 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
103: 22: found reward = -0.5  from time 97.0
   103: 22 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 23 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 24 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
   103: 25 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
103: 26: found reward = -0.5  from time 101.0
   103: 26 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
103: 27: found reward = 0.5  from time 98.0
```

```
103: 27: found reward = -0.5  from time 100.0
  103: 27 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  103: 28 - State 17  Action 5  Reward 0.5  newState 59  from 28
103: 29: found reward = 0.5  from time 101.0
  103: 29 - State 17  Action 5  Reward 0.5  newState 59  from 28   17 5 0.5 59 28
  104: 15 - State 59  Action 9  Reward 0.5  newState 59  from 15
  104: 15 - State 17  Action 1  Reward -0.5  newState 50  from 22   17 1 -0.5 50 22
  104: 15 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 15 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 15 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 15 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29
  104: 16 - State 59  Action 9  Reward 0.5  newState 59  from 15   59 9 0.5 59 15
  104: 16 - State 17  Action 1  Reward -0.5  newState 50  from 22   17 1 -0.5 50 22
  104: 16 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 16 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 16 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 16 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29
  104: 17 - State 59  Action 9  Reward 0.5  newState 59  from 15   59 9 0.5 59 15
  104: 17 - State 17  Action 1  Reward -0.5  newState 50  from 22   17 1 -0.5 50 22
  104: 17 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 17 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 17 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 17 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29
  104: 18 - State 59  Action 9  Reward 0.5  newState 59  from 15   59 9 0.5 59 15
  104: 18 - State 17  Action 1  Reward -0.5  newState 50  from 22   17 1 -0.5 50 22
  104: 18 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 18 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 18 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 18 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29
  104: 19 - State 59  Action 9  Reward 0.5  newState 59  from 15   59 9 0.5 59 15
  104: 19 - State 17  Action 1  Reward -0.5  newState 50  from 22   17 1 -0.5 50 22
  104: 19 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 19 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 19 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 19 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29
  104: 20 - State 59  Action 9  Reward 0.5  newState 59  from 15   59 9 0.5 59 15
  104: 20 - State 17  Action 1  Reward -0.5  newState 50  from 22   17 1 -0.5 50 22
  104: 20 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 20 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 20 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 20 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29
  104: 21 - State 59  Action 9  Reward 0.5  newState 59  from 15   59 9 0.5 59 15
  104: 21 - State 17  Action 1  Reward -0.5  newState 50  from 22   17 1 -0.5 50 22
  104: 21 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 21 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 21 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 21 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29
  104: 22 - State 59  Action 9  Reward 0.5  newState 59  from 15   59 9 0.5 59 15
  104: 22 - State 17  Action 1  Reward -0.5  newState 50  from 22
  104: 22 - State 62  Action 10  Reward -0.5  newState 53  from 26   62 10 -0.5 53 26
  104: 22 - State 50  Action 4  Reward 0.5  newState 53  from 27   50 4 0.5 53 27
  104: 22 - State 50  Action 5  Reward -0.5  newState 53  from 27   50 5 -0.5 53 27
  104: 22 - State 59  Action 6  Reward 0.5  newState 59  from 29   59 6 0.5 59 29

                                          :
                                          :
```

# APPENDIX C

# PUBLISHED PAPERS

The following pages Contain information on all published Journal Articles and Conference papers arising from this research. The Titles, Details of publications and Abstracts are included.

## C.1 Abstracts

**Title**: Effects of Communication in Cooperative Q-Learning

**Authors**: Darbyshire, P. and Wang, D.

**Publication**: International Journal of Innovative Computing, Information and Control (IJICIC),

**Details** : Vol 6, No. 5, pp 2113-2126, May 2010, ISSN 1349-4198

**Abstract**

Reinforcement learning has been utilized to investigate the distributed learning problem in many different multi-agent, team-based scenarios. Invariably, using a distributed learning approach by allowing agents to exchange what they have learned, by either direct communication or by episodic exchanges, the team does better in terms of achieving its goals. The cause-effect relationship between communication and improved performance has been studied in a number of different team based scenarios; however, it is difficult to find a detailed analysis of the effect the communication has as a function of time on the elements of the simulation. Of particular interest is the effect on the learning of agents when part of a communicating multi-agent system. Studies show the effects of cooperative learning in the form of accelerated group learning rates; but it is difficult to pinpoint through the literature exactly how this observed effect is realized within individual agents over time. This paper investigates the effect of communication as a function of time on a multi-agent simulation based on a military distillation, utilizing a modified Q-learning algorithm.

**Title**: Using XML to Help Isolate Software Systems and Agents from Change Due to Communications

**Authors**: Darbyshire, P.

**Publication**: Journal of Business Systems, Governance and Ethics (JBSGE)

**Details** : Vol 1 No. 4, 2006, pp 59-68, ISSN 1833-4318

**Abstract**

Development and research into distributed and agent based systems has grown enormously over the last few years, and the number of practical applications for such systems has grown along with it as the technology and infrastructure improves to accommodate such systems. As with all systems, evolution and change is inevitable, but with the growth of distributed systems and the Service Oriented Architecture, we have another dimension of change we need to consider; that of communication. The importance of the role of communication between these systems has been highlighted by many researchers, particularly for multi- agent systems and for distributed communicating agents. But the form of such communication often remains a mystery. Communication aspects are often dependent on other factors within an architectural framework, particularly the data. In order to reduce unnecessary changes to the communication aspects of a system, we need to insulate the communication as much as possible from consequential change effected by architectural other framework elements. A message system using an XML-type syntax is more extensible and adaptable for use in a changing environment. It helps to isolate the communication from the structure and content of the message, thereby reducing consequential change. This paper discusses the use of XML for the construction of agent-based messages, and presents a simple approach for the deconstruction of messages by receiving agents.

**Title**: Learning to Survive: Increased Learning Rates by Communication in a Multi-agent System

**Authors**: Darbyshire, P. and Wang, D.

**Publication**: Lecture Notes in Artificial Intelligence, Presented at AI03

**Details** : (LNAI 2903), Springer, ISBN 3-540-20646-9 pp 601-611, 2003

**Abstract**

In a hostile multi-agent environment, a team of learning agents utilizing a reinforcement-learning paradigm may not learn at a sufficient rate for the agents to adapt and survive. If we can increase the learning rate of the individual agents, they may learn earlier to select more optimal actions and hence in- crease the chance of survival. Implementing a simple communication structure between the agents and allowing them to communicate their learning events with team members at each time step considerably increases the learning rate. This increased learning rate can significantly improve the team's chances of survival in the agent's environment, but alone is not a guarantee of better performance. The type of information communicated also plays a crucial role in determining increased survival chances. This paper reports on some experimental results from simulating two teams of combative agents, one utilizing reinforcement learning as a control paradigm using communication to increase the measured learning rate.

**Title**: Effects of Communication on Group Learning Rates in a Multi-agent Environment

**Authors**: Darbyshire, P.

**Publication**: Advances in Complex Systems, Special Issue: Agent-Based Approaches in Complex Systems

**Details** : Vol 6 No 3, September 2003, pp 405-426, ISSN 0219-5259

**Abstract**

Distillations utilize multi-agent based modeling and simulation techniques to study warfare as a complex adaptive system at the conceptual level. The focus is placed on the interactions between the agents to facilitate study of cause and effect between individual interactions and overall system behaviour. Current distillations don't utilize machine-learning techniques to model the cognitive abilities of individual combatants but employ agent control paradigms to represent agents as highly instinctual entities. For a team of agents implementing a reinforcement-learning paradigm, the rate of learning is not sufficient for agents to adapt to this hostile environment. However, by allowing the agents to communicate their respective rewards for actions performed as the simulation progresses, the rate of learning can be increased sufficiently to significantly increase the teams chances of survival. This paper presents the results of trials to measure the success of a team-based approach to the reinforcement-learning problem in a distillation, using reward communication to increase learning rates.

**Title**: Modeling Agent Communication in a Complex System as a Neural Net

**Authors**: Darbyshire, P.

**Publication**: Lecture Notes in Engineering and Computer Science

**Details** : Intentional Association of Engineers, Vol 2194 Issue 1, 2011, pp 1032-1037, ISSN 2078-0958

**Abstract**

Communication between cooperating agents has been shown in research to enhance the learning process of agents and their ability to complete tasks successfully in a group environment. However, an environment where we have many independent interacting components (agents), with each affecting the decision processes of the others, is essentially a complex system. Studying the behavior of the inter-agent communication in a complex system is difficult, particularly as the interaction between the agents may also affect the communication type and level. However, modeling the agent communication as a Neural Net may provide a convenient way to view such communication and further research it's effect on the learning process.

*Note: also published in IEEE, Proceedings of the 2010 International Conference on Modeling, Simulation and Control – see below.*

**Title**: Using a Neural Net to Model Agent Communication in a Complex System

**Authors**: Darbyshire, P.

**Publication**: IEEE, Proceedings of the 2010 International Conference on Modeling, Simulation and Control

**Details** : Cairo, Egypt, November 2010, pp 454-458, ISBN 978-1-4244- 8823-0

**Abstract**

Communication between cooperating agents has been shown in research to enhance the learning process of agents and their ability to complete tasks successfully in a group environment. However, an environment where we have many independent interacting components (agents), with each affecting the decision processes of the others, is essentially a complex system. Studying the behavior of the inter-agent communication in a complex system is difficult, particularly as the interaction between the agents may also affect the communication type and level. However, modeling the agent communication as a Neural Net may provide a convenient way to view such communication and further research it's effect on the learning process.

**Title**: What did they Learn?: the Effect on Learned Behavior of Increased Learning Rates in a Multi-agent System

**Authors**: Darbyshire, P., Wang, D.

**Publication**: 7th Asia-Pacific Conference on Complex Systems

**Details** : Carins Australia, Dec 2004. pp 476-488 , ISBN 1-876674-96-2

**Abstract**

In a simulated complex system representing a military distillation, the environment is extremely hostile to competing agent teams. Unless individual agents learn to survive they may die, thus significantly affecting their team's chances of success. For a group of learning agents, we can increase the learning rate of the agents by implementing communication between team members and thus increase the teams survival and success chances. While the effect of communication on learning rates for individual agents, and the team is significant, it is difficult to pinpoint any one simulation metric resulting from this which points to the observable difference in outcomes. This would seem to indicate a butterfly effect in evidence. However a significant difference in learning rates should translate to some quantifiable change elsewhere. Rather than look for differences in simulation metrics we need to examine what the agents have learned. By examining the differences in the learned behavior of agents in the simulation at various points we can observe the impact of the increased learning rate. This paper examines the impact of increased learning rates facilitated by agent communication on the learned behavior of agents controlled by a reinforcement-learning paradigm. We examine the knowledge landscape from the action-value matrices and compare the learned behavior of different groups of agents at various times throughout the simulation trials.

**Title**: Using XML for Simple Hierarchical Communication between Agents

**Authors**: Darbyshire, P.

**Publication**: International Resource Management Association (IRMA) 2004

**Details** : New Orleans, USA, May 2004, pp 686-689, ISBN 159140-279-4

**Abstract**

Research in agent-based systems has grown enormously over the last few years, and the number of practical applications for agent systems has grown along with it as the technology and infrastructure improves to accommodate such systems. The importance of the role of communication between the agents has also been highlighted by many researchers, particularly for multi-agent systems and for distributed communicating agents. But the form of agent communication often remains a mystery. Agent communication languages and standards are being developed however the dust hasn't really settled yet, and many implementations use ad-hoc techniques. XML provides an excellent form for agent communication by facilitating the construction of hierarchical message structures. Many of the practicalities for implementation of message recognition can be overcome by the utilization of existing libraries for XML parsing. A message system using an XML-type syntax is more extensible and adaptable for use in a changing environment. This paper discusses the use of XML of the construction of agent-based messages, and presents a simple approach for the deconstruction of messages by receiving agents.

**Title**: Bringing Intelligence to the Battlefield

**Authors**: Darbyshire, P., McKay, B.

**Publication**: Modelling and Simulation (MS'02) 2002

**Details** : Melbourne, Australia, 2002, ISBN 1-86272-617-5

**Abstract**

Agents and agent technology has been with us for some time, but in particular, the use of agent-based paradigms in simulations are a fairly recent development. Multi-agent simulations are becoming increasingly popular due in part to the sophistication of Object Oriented techniques for constructing them, but also because of their ability to model Complex Adaptive Systems. Distillations, now used in military simulation, represent a step from the traditional approach to one based on multi-agent simulations, where the focus is on the combatants and their behaviour. Yet, to effectively model combat scenarios, we also need to model the cognitive abilities of the combatants. This paper reports on the implementation of learning techniques for agents in a distillation used to explore battlefield behaviour.

**Title**: Using Communication to Increase Learning in a Hostile Multi-Agent Environment

**Authors**: Darbyshire, P., McKay, B.

**Publication**: Complex Systems 2002

**Details** : Tokyo, Japan, 2002

**Abstract**

Distillations utilize multi-agent based modelling and simulation techniques to study warfare as a complex adaptive system at the conceptual level. The focus is placed on the interactions between the agents to facilitate study of cause and effect between individual interactions and overall system behaviour. Current distillations don't utilize machine-learning techniques to model the cognitive abilities of individual combatants but employ agent control paradigms to represent agents as highly instinctual entities. For a team of agents implementing a reinforcement-learning paradigm, the rate of learning is not sufficient for agents to adapt to this hostile environment. However, by allowing the agents to communicate their respective rewards for actions performed as the simulation progresses, the rate of learning can be increased sufficiently to significantly increase the teams chances of survival. This paper presents the results of trials to measure the success of a team-based approach to the reinforcement-learning problem in a distillation, using reward communication to increase learning rates.

**Title**: A Prototype Design for Studying Emergent Battlefield Behaviour through Multi-Agent Simulation

**Authors**: Darbyshire, P., Abbass, H., Barlow, M., McKay, B**.**

**Publication**: The 4th Japan- Australia Joint Workshop on Intelligent and Evolutionary Systems

**Details** : Hayama-machi, Japan, 2000

**Abstract**

Over the past few years, Multi-Agent Systems have become an important tool in computer science, particularly in the area of Artificial Intelligence and Simulation. Simulation is one of the major areas of application for multi-agent systems, bringing a new dimension to simulation by allowing us to directly represent individuals and their interaction. In fact it is the emphasis on the interactions between agents that allow us to study the resulting emergent behaviour, which distinguishes multi-agent systems from other approaches. This paper describes a research project at the Australian Defence Force Academy that provides a framework for simulating a land combat situation using a multi-agent system. Emphasis is placed on the individual learning by agents, as well as group learning. By using a multi-agent system the emergent behaviour between the agents will become the main focus of the research.