

2004

Optimizing combat capabilities by modeling combat as a complex adaptive system

Steven Mains

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#), [Industrial Engineering Commons](#), and the [Operational Research Commons](#)

Recommended Citation

Mains, Steven, "Optimizing combat capabilities by modeling combat as a complex adaptive system" (2004). *Dissertations, Theses, and Masters Projects*. Paper 1539623454.

<https://dx.doi.org/doi:10.21220/s2-5bpz-c473>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

Optimizing Combat Capabilities by Modeling Combat as a Complex
Adaptive System

A Dissertation

Presented to

The Faculty of the Department of Computer Science

The College of William and Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree of

Doctor of Philosophy

by

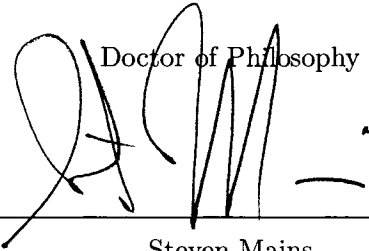
Steven Mains

2004

APPROVAL SHEET

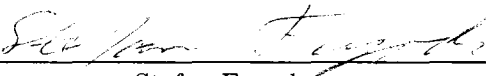
This dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy



Steven Mains

Approved, November 2004



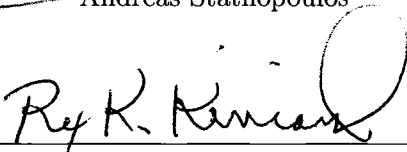
Stefan Feyock
Thesis Advisor



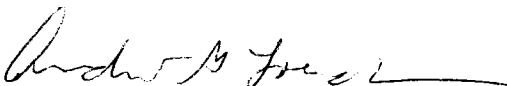
Weizhen Mao



Andreas Stathopoulos



Rex Kincaid
Department of Mathematics



Andrew Loerch
George Mason University

To Maureen, Peter and Caroline...

Table of Contents

Acknowledgments	ix
List of Tables	xi
List of Figures	xii
Abstract	xiii
1 Introduction	2
1.1 Statement of the Problem	2
1.1.1 Budgeting as a Balancing Act	2
1.1.2 The Force Development Process	3
1.1.3 Non-linearity of Combat	6
1.1.4 The Tyranny of Multi-dimensionality	8
1.2 Requirements For Combat Development Models	10
1.3 Proposed Approach	11
1.4 Contribution	12
1.5 Classification Concerns	13

2	Relevant Work	14
2.1	Attempts to Solve the Force Development Problem	14
2.1.1	Modeling Origins	14
2.1.2	Simplification by Exclusion	18
2.2	Current Combat Models	19
2.2.1	Janus	19
2.2.2	Modular Semi-Automated Forces (MODSAF)	21
2.2.3	Shortcomings	22
3	Modeling Combat as a Complex Adaptive System	24
3.1	Definition of a Complex Adaptive System	25
3.2	Applicability	30
3.2.1	Multiple Agents	30
3.2.2	Adaptation	31
3.2.3	Self-organization	31
3.2.4	Non-linearity	32
3.3	Model Description	32
3.3.1	General	33
3.3.2	Conceptual Model	34
3.3.3	Specification Model	35
3.3.3.1	Physical States	35
3.3.3.2	Tactical States	37
3.3.4	Computational Model	39

3.4	Verification	45
3.5	Validation	46
4	Searching the Space	47
4.1	Characterization of the Fitness Space	48
4.2	Available Approaches	53
4.2.1	Derivative Approaches	53
4.2.2	Frequency Domain Method	54
4.2.3	Differential Qualitative Analysis	54
4.2.4	Response Surface Methodology	55
4.2.5	Evolutionary (Derivative Free) Approaches	56
4.2.6	Evolution Strategies	57
4.2.7	Evolutionary Programming	57
4.2.8	Genetic Algorithms	58
4.2.9	Genetic Programming	59
4.3	Artificial Life	60
4.3.1	Definition	60
4.3.2	Evolving Structures	62
4.3.3	Evolving Rules	62
4.3.4	Evolving Both Structures and Rules	63
4.4	Ant Colony Simulations	63
4.5	Enhancement Through Co-evolution	66
4.6	Co-evolutionary Theory	70

4.6.1	Increased Exploitation	71
4.6.2	Increased Exploration	75
4.7	The Genetic Algorithm Approach	77
4.7.1	General Approach	77
4.7.2	Selection of Solutions	78
5	Modeling Results	80
5.1	Improved Solutions	81
5.2	Increased Exploration	82
5.3	Appropriate Solutions	84
5.3.1	General	85
5.3.2	Sights	86
5.3.3	Armor Protection	87
5.3.4	Weapon Systems	87
5.3.5	Engines	89
5.3.6	Tactics	90
5.4	Conclusions	91
6	The Value of Information	94
6.1	Approach	96
6.2	Results	97
6.2.1	Increased Fitness	97
6.2.2	Information as a Substitute for Capabilities	99
6.3	Assessment of the Value of Information	100

6.4	Assessment of the Validity of the Approach	101
7	Further Work	102
8	Conclusion	104
8.1	Intent of the Dissertation	104
8.2	Restatement of the Problem and Approach	106
8.3	Results	108
8.4	Measurement of the Value of Information	109
8.5	Implications for Future Work	109
Appendix A	Code for the Agent-based Model	111
Appendix B	Code for the Co-evolutionary Genetic Algorithm	228
Appendix C	Chromosome Definitions	248
Bibliography		253
Vita		260

ACKNOWLEDGMENTS

I could not have completed this effort without the assistance and direction of my co-advisors, Professors Stefan Feyock and Rex Kincaid, at the College of William and Mary. They have given me the freedom to explore while always being ready to provide encouragement and advice. I am indebted to them for their support, working around many deployments. I would like to thank the Professors and Staff of the Computer Science Department, and especially my committee and the Computational Operations Research Program, for their encouragement and guidance.

I am indebted to my classmates, who taught me more than anyone at the College.

Most of all, however, I want to thank the men and women I have served with in 24 years in the Army for their support and their contributions to allowing the pursuit of happiness that we all enjoy.

List of Tables

3.1	Typical Rule Set Tuples.	38
4.1	Probability of Survival for d -order schemata	75
5.1	Highest Fitness Found.	82
5.2	Objective Value Range of Solutions Found.	83
5.3	Sight System Capabilities and Cost	86
5.4	Selected Weapon Systems Capabilities and Cost	88
5.5	Capabilities and Cost for Non-selected Weapons	89
5.6	Engine Cost and Parameters	89
5.7	Rule Predominance	90
6.1	Previous Runs v. Information Enhanced Excursion	97
6.2	Fit Solutions Comparison	98

6.3	Comparison of Information-Enhanced Solutions to Standard Information Level	98
C.1	Tank Physical Gene Definitions.	248
C.2	Tank Tactical Gene Definitions.	249
C.3	Tank Tactical Gene Definitions (cont'd).	250
C.4	Artillery Physical Gene Definitions.	250
C.5	Artillery Tactical Gene Definitions.	251
C.6	Artillery Tactical Gene Definitions (cont'd).	252

List of Figures

4.1	Two-dimensional Landscape Representation.	51
4.2	Three-dimensional Landscape Representation.	52
4.3	Close-up of a Portion of the Three-dimensional Landscape Representation.	53
4.4	Comparison of One-step Range.	76
5.1	Effect of Overemphasis on Remaining Near Friendlies.	92

ABSTRACT

Procuring combat systems in the Department of Defense is a balancing act where many variables, only some under control of the department, shift simultaneously. Technology changes non-linearly, providing new opportunities and new challenges to the existing and potential force. Money available changes year over year to fit into the overall US Government budget. Numbers of employees change through political demands rather than by cost-effectiveness considerations. The intent is to provide the best mix of equipment to field the best force against an expected enemy while maintaining adequate capability against the unexpected. Confounding this desire is the inability of current simulations to dynamically model changing capabilities and the very large universe of potential combinations of equipment and tactics.

The problem can be characterized as a stochastic, mixed-integer, non-linear optimization problem. This dissertation proposes to combine an agent-based model developed to test solutions that constitute both equipment capabilities and tactics with a co-evolutionary genetic algorithm to search this hyper-dimensional solution space. In the process, the dissertation develops the theoretical underpinning for using agent-based simulations to model combat. It also provides the theoretical basis for improvement of search effectiveness by co-evolving multiple systems simultaneously, which increases exploitation of good schemata and widens exploration of new schemata. Further, it demonstrates the effectiveness of using agent-based models and co-evolution in this application confirming the theoretical results.

An open research issue is the value of increased information in a system. This dissertation uses the combination of an agent-based model with a co-evolutionary genetic algorithm to explore the value added by increasing information in a system. The result was an increased number of fit solutions, rather than an increase in the fitness of the best solutions. Formerly unfit solutions were improved by increasing the information available making them competitive with the most fit solutions whereas already fit solutions were not improved.

Optimizing Combat Capabilities by Modeling Combat as a Complex
Adaptive System

Chapter 1

Introduction

The most extensive computation known has been conducted over the last billion years on a planet-wide scale: it is the evolution of life. The power of this computation is illustrated by the complexity and beauty of its crowning achievement, the human brain.

–David Rogers

1.1 Statement of the Problem

1.1.1 Budgeting as a Balancing Act

Procuring combat systems in the Department of Defense is a balancing act where many variables, only some under control of the department, shift simultaneously. Technology changes non-linearly, providing new opportunities and new challenges to the existing and potential force. Money available changes year over year to fit into the overall US Government budget. Numbers of employees (soldiers, sailors, Department civilians, etc.) change as a

result of political forces¹ rather than by an analysis of what is most cost-effective. The intent is to provide the best mix of equipment and organizations in order to field the best force against an expected enemy while maintaining a capability against the unexpected. Armies seldom get it completely right when planning for an adversary. As British Historian Michael Howard puts it, we don't have to get it completely right but we must get it "less wrong than the adversary" [29].

To find the best mix of technology and combat processes (tactics) in this very dynamic environment, the Department of Defense has developed the Force Development process. As will be seen, this system is hampered by a lack of modeling tools capable of capturing the complexity of combat and a way to find the best set of technology and processes in an extremely large solution space. This dissertation proposes a solution to these shortcomings.

1.1.2 The Force Development Process

The Force Development process consists of both top-down and bottom-up processes. The top-down process consists of annual mission, capability and budget guidance developed by the administration with input from Congress and passed through the Secretary of Defense and Joint Chiefs of Staff.² The Joint Chiefs model alternative force structure and capability options against projected threats and issue guidance to the services regarding their capability requirements and budget limitations.

¹"End strength," or the number of troops in each service, is regulated by law. The services have input into the number, but actual control over the number resides in the Congress. Ideally, upward changes in the number of forces would be accompanied by corresponding funds to pay for the accession, training, salary and benefits for those extra employees, but in practice increases may not be accompanied by funding.

²The Chairman of the Joint Chiefs of Staff and the uniformed heads of the Army, Navy, Marines and Air Force.

The bottom-up component is the result of changing threat capabilities, experience gained in real operations (such as Operation Iraqi Freedom or Bosnia), lessons learned at training centers such as the National Training Center (NTC)³ and technological breakthroughs. As the threat evolves, services develop new ways to counter the changes. Technology provides new opportunities that the services attempt to incorporate. The Army has defined the changes to be in one of five domains: Doctrine, Training, Leader Development, Organization and Materiel [1].⁴ These domains are roughly listed in order of increasing cost. Changing doctrine,⁵ aside from a nominal cost to change some field manuals and institutional training packages, is essentially without cost. Training and Leader Development⁶ changes are usually more expensive, as they may require new training devices, but the cost pales in comparison to organizational changes, which may require changing the number of people in the organization. This may require redistribution and retraining of people, as well as recruiting and training more people with new skill sets.

The most expensive domain in terms of direct cost is Materiel, which entails developing, buying and fielding new systems. Each new system requires extensive engineering and testing to ensure that it can withstand the rigors of combat and operate in environments as diverse as high mountains and barren deserts. There is a non-trivial cost to establish a production facility at the beginning of a system's cost learning curve, particularly for

³At Fort Irwin, CA. The NTC is a very large desert training facility where units up to Brigade-level (3-4 Battalions or approximately 1500 soldiers) can operate a force-on-force exercise against an opposing force. Each vehicle on both sides is instrumented to record vehicle location and actions over time so that the reactions can be evaluated and to identify training shortcomings.

⁴Often these are abbreviated as DTLOM

⁵Doctrine is defined as the "rule" that a force uses to defeat an enemy and to accomplish its mission. For our purposes I will use the terms doctrine and tactics synonymously

⁶Meaning changing the way we assess, train and mature our leaders in institutional schools and unit training.

resource-intensive systems like 70-ton tanks that have no civilian counterparts. The units receiving new equipment must be trained to use the equipment at not inconsiderable cost and their old equipment must be demilitarized.⁷

Adding to the cost at each higher-level domain is the fact that a change at one domain causes changes to the lesser-cost domains as well. A change of organization could require not only more equipment and soldiers. It could also require development of more leaders to command the new organizations. At the very least, it requires new training and doctrine.

Each service conducts extensive computer simulations followed by live testing of proposed solutions in an effort to find the least cost set of domain changes in response to requirements. To complement this process, each service staff balances solutions with the guidance received from the Joint Chiefs in an effort to develop coherent and effective one- and five-year spending plans. These plans, along with the assumptions and models used in their preparation, are in turn reviewed by the Secretary of Defense, the Joint Chiefs, and Congress.⁸ Despite the amount of effort put into this process, analysis of combat systems is inherently complex due to the non-linearities of combat and the huge number of alternatives available to decision-makers.

⁷Usually the active forces receive new equipment first, so disposal includes moving their older displaced equipment to the reserve forces and subsequent disposal of the reserves' equipment. Due to the amount and type of reserve forces, there may be a multiple cascade of equipment requiring movement to lower priority units and disposal. This cascading cost is included when considering new pieces of equipment and in part accounts for the high price tag of weapon systems.

⁸Not to mention every defense contractor whose system "lost" in the budget process and was not funded.

1.1.3 Non-linearity of Combat

Combat is a peculiarly human phenomenon. The outcome is the collective result of the individual decisions of a large number of actors and their interaction with their fellow combatants, the environment, the enemy, and their equipment. These “agents” act⁹ based on their individual understanding of the mission, their knowledge of tactics and the perceived threat. Although Generals can send orders to the front, or even deliver those orders personally in order to influence the action by their personal leadership, the individual agent – the tank commander¹⁰ or the infantryman – makes the final decision as to what action it takes.¹¹ The action of these agents can be intense, chaotic and, to an observer, inscrutable, but it results in identifiable macro-level behavior in the overall system. Like the stock market, which is often described anthropomorphically as being “jittery” or “advancing,” defenses can be said to “buckle” or “strengthen” while assaults “sputter” or “overwhelm.”

The interaction of the agents is non-linear because of the concept of positive and negative feedback [3][4][19]. Positive feedback provides rewards based on the results of actions. Negative feedback penalizes the agent. In combat, as an attack progresses successfully, the attackers become more confident of success and press the attack harder. Success provides

⁹For my purposes I will discuss agents here without a formal definition, which will come later. The concept is evident enough for the purpose of description of the overall combat environment. For these purposes, I define tank agents as the entire tank system, which includes the crew and the combat equipment as if they are one entity. The same will be done with other types of agents.

¹⁰It is important to define tank commander as the sergeant or officer that commands an individual tank. The term tank commander is sometimes used by historians when discussing skilled leaders of large tank formations, like Rommel or Patton, to distinguish them from commanders who did not grasp how to use the new tanks effectively. Tank commander is the official Army term for one who commands a single tank and will be used throughout the dissertation as such.

¹¹Even Field Marshall Erwin Rommel found himself leading a one-man charge after ordering a squad to attack in Italy in World War I [75]. The “agents” in the squad decided the risk was too great and remained under cover.

positive feedback, which increases the likelihood of further success. Similarly, local failure instills and reinforces expectations of ultimate failure, thus increasing the likelihood of failure.

Individual events and small units can have disproportionate effects. An example is the experience of the French Army near Sedan, France at the outset of World War II [15]. On May 13, 1940 inaccurate reports of the presence of German units nearby caused a French artillery battalion to reposition. As the unit moved, other units surmised that a retreat was occurring. Within two hours, the entire artillery support for the French 55th Division was in headlong retreat, allowing the German attackers a relatively easy victory in that area.¹²

Similarly, only 10 of the 117 German divisions that attacked France in 1940 ever broke free from their railway-based logistics systems. Those few divisions, even though they were equipped with equipment that was at best equal, and in most cases inferior, to that of the French and British forces, caused most of the collapse of the Allied Forces [76].

Effects of new capabilities¹³ on operations are also non-linear. In a large army, introduction of a small quantity of some new piece of equipment generally has no effect on overall effectiveness. Once a critical mass is reached, however, the new equipment improves combat capability until reaching a point of diminishing returns where more of a particular type of equipment is not helpful and may even become detrimental to effectiveness.¹⁴ Chapter 2 will review the current models and their shortcomings when addressing these non-linearities.

¹²This is not to imply that the German Army would not have been successful in their invasion of France without this incident. It is merely one example of non-linear interactions in military operations.

¹³This might be a larger cannon, improved speed or reduced visual signature.

¹⁴At some point a new capability would begin to divert resources from other higher-payoff activities and the effectiveness curve would turn downward.

Chapter 3 will discuss how combat modeling can change to account for these non-linearities.

1.1.4 The Tyranny of Multi-dimensionality

A problem alluded to above is the combinatorial complexity caused by the very large number of solutions possible when even a small number of systems and capabilities are considered. This exponential explosion of possible combinations forces the investigator to limit the number of candidate solutions either arbitrarily or through some sort of “Delphi Technique” where subject matter experts select the most promising of the systems under study to keep the number of alternative solutions manageable.

To use a very simple example, consider the interactions between three types of weapon systems with five system capabilities each. Say the three systems are a tank, an infantry squad and an artillery piece and the five capabilities are speed, weapon range, vision distance, survivability¹⁵ and fuel consumption. These parameters are continuous values, which results in an infinite number of combinations unless discretized into, say, five values. Even after such simplification of the problem by restricting the options, the solution space still consists of 10^{10} combinations.

This solution space, however, is not fully representative of the actual problem. It assumes that tactics are fixed across the solution space and that the numbers of equipment are static. Cost constraints should, as systems become more capable (and expensive), decrease their numbers in a force, inducing changes in the employment of that force. Clearly the tactics

¹⁵Quantified as millimeters of armor. More armor results in an increased probability of surviving a hit from an enemy weapon.

would be very different if a force consisted of a large number of tanks and few infantry (as the US forces in the Gulf War and Operation Iraqi Freedom) or large infantry formations (as the Chinese Army in 1952). Including the different tactics that could be used with the different types and amount of equipment would increase the dimensionality of the solution space from its already daunting size.

Determining the appropriate tactics is surprisingly difficult without extensive analysis especially when the capabilities under study are novel. Often the best tactics are not obvious or are unacceptable to decision makers because of unconventionality or because they challenge the status quo. A historical example will suffice to illuminate this point. Brigadier General William “Billy” Mitchell was a great air power enthusiast in World War I. After the war, he championed many uses for air power, but his most famous is the proposed use of aircraft to defeat battleships. He theorized that an air fleet, operating from land, could defeat a naval fleet.

He pressed for tests of his concept, which were staunchly opposed by the US Navy. Fear of increasing inter-service rivalry caused even other Army Generals to support Mitchell’s efforts only tepidly. He was finally able to get permission to test his concept on captured German battleships and excess US battleships due to be scuttled off Norfolk, VA in 1921. Mitchell’s pilots succeeded in sinking one of the largest and most heavily armored battleships in the world, the *Ostfriesland*, followed by the battleships USS Alabama, Virginia and New Jersey [26]. In doing so, Mitchell set conventional wisdom on its head and changed naval tactics forever. To truly develop appropriate tactics, they must be developed alongside the capabilities that they complement, rather than as an input variable or as an afterthought.

Chapter 4 will explore how to search this multidimensional solution space.

1.2 Requirements For Combat Development Models

From this discussion, four requirements for combat models can be discerned. They must account for the non-linearity of combat. They must account for a myriad of potential tactics to best support the equipment capabilities. They must be able to search large solution spaces and they must do it quickly enough to be useful. The historical examples presented emphasize the non-linearity. They clearly indicate that the effect of forces on the battlefield can be disproportionate to the size and firepower of the force. This is because when a force surprises the enemy, employs an unexpected weapon or tactic, is more skillfully led or simply is more determined than the enemy, it will have more success than would normally be expected. It is not sufficient to count force strengths and attempt to draw conclusions from an expected value derived from the force ratios. Although Werner Heisenberg said “The equation knows best,” [25] for a combat model, there is no closed form equation.

The requirement for multiple tactics based on the equipment mix raises another requirement for combat models – there must be a way to vary the tactics of the weapon systems based on capabilities and numbers automatically in order to compare the best tactics for one equipment mix to the best tactics for an alternate equipment mix.

The combinatorial complexity of the solution set requires searching extremely large solution spaces where trade-offs in and between DOTLM domains must be made in order to determine the best force for a given situation and against a particular enemy.

The stochastic nature of the interactions between the agents indicates that each alternative must be run enough times to get some statistical certainty that one alternative is better than another. This increases the computing time required to search the solution space. The simulation must necessarily be fast¹⁶ and provide meaningful measures of combat capability in order to support weapons procurement decisions.

1.3 Proposed Approach

The problem under study can be characterized as a mixed-integer, non-linear optimization problem that lacks a closed-form representation but has an extremely large solution set. To address this problem, the research described in this dissertation uses simulation optimization combining an agent-based model to determine fitness of a selected solution with a genetic algorithm to choose from the very large solution space. The model will consist of tank and artillery agents that follow a rule set that can vary between generations along with the capabilities of the systems. The agent's capabilities and tactics will be represented as a binary string formed into a "chromosome." This chromosome defines the universe of alternatives, which allows a genetic algorithm to search the solution space. To enhance the genetic algorithm, the two systems will cooperatively coevolve. The overall fitness of the set agent types, which follow the rule sets encoded in their chromosome along with the physical characteristics, defines the fitness of the solution. Contribution of each type of agent to the overall fitness will not be determined. Solutions will combine and continue to

¹⁶Fast is, of course, relative. The simulation must support searching the solution space sufficiently quickly to be of use to a decision maker. This means that the solution space must be searched in a matter of days, at most. Longer than that would make the model too cumbersome to use in any but the most deliberate analyses of alternatives.

the next generation or be eliminated from the solution sample based only on the overall fitness. Chapter 5 documents the results of this approach.

1.4 Contribution

This dissertation provides contributions to the *corpus scientia* in a number of areas. First it proposes a model that adapts both system capabilities and rule sets to solve a real-world problem. It provides proof that co-evolutionary genetic algorithms are superior to evolutionary genetic algorithms, a proposition that has been shown empirically, but has never been rigorously explained. In developing the model, this dissertation proposes a standard categorization technique for fitness landscapes that can be used as a first check as to whether a problem is suited to a particular solution method. In the discussion of Complex Adaptive Systems, it synthesizes a definition for them from the many that have been proposed and rigorously compares the characteristics of combat to the definition. Lastly, much has been written about the ability to measure the value of information in a physical system, but the very issues of non-linearity and combinatorial complexity that have limited combat simulations have limited efforts to measure information. As a result, much of the measurement of the value of information has been inferential rather than direct. To test the value of the modeling process to measure information directly, an excursion was run and conclusions drawn.

1.5 Classification Concerns

This dissertation is completely unclassified. All data used in this modeling effort was either gathered from open source web sites, such as the Federation of American Scientists, or, when not available, estimated by the author. The purpose was to explore an approach rather than develop an empirical answer to the problem. The model can be easily recoded with the classified data relevant to actual systems under study to develop a solution.

Chapter 2

Relevant Work

To seek out the best through the whole Union, we must resort to the information which from the best of men, acting disinterestedly and with the purest motives, is sometimes incorrect.

–Thomas Jefferson

2.1 Attempts to Solve the Force Development Problem

The problem described is not novel. It has been an area of interest for over 5000 years. This chapter will review the efforts to solve two of the problems outlined in the previous chapter – modeling combat dynamically and searching the large universe of solutions.

2.1.1 Modeling Origins

Models or simulations typically are used when the system under study is unavailable or too expensive to study directly in operation [6]. Likewise, simulation is often the only way to model a system that is dynamic and evolves based on the attributes and actions

of the participants. Combat fits these criteria precisely. It requires a huge investment in personnel, land, fuel and ammunition to conduct a meaningful, non-lethal wargame (i.e. a live simulation) with actual troops and equipment. It is much more cost-effective to use other (non-live) simulation techniques to develop equipment and tactics before fully testing the concept with live troops in a realistic wargame. Confounding the problem of conducting a realistic wargame against a projected enemy is that the very army or armies that would be most useful as an opposing force would be unwilling to lend their expertise, manpower and equipment to such an enterprise.

Before computers, physical representations of armies were used in simulations. Miniature soldiers have been found in the burial effects of kings in ancient Sumeria and Egypt indicating that combat modeling may have occurred earlier, but combat modeling really can be said to have started when the Chinese General *Sun Tsu* developed a game called *Wei Hai* about 5000 years ago [67]. In the subsequent centuries, modeling was primarily in the form of a two-sided board game with rules for movement and tactics, but no fixed rules to adjudicate losses. In the Civil War Abraham Lincoln recognized that successful attacks generally required a 3:1 advantage over an entrenched defender [16] and a simple, quantitative rule-of-thumb for determining relative advantage was born.

This rule sufficed as a guide until the 1880's when the Prussian Army devised *Kriegspiel* – a board game played on a grand scale, filling the central square of the *Kriegsakademie* [67]. Students were assigned to work out the rate of movement for each unit represented and move those unit pieces tactically across a grid. The game board was configured with terrain data in each square. A group of “umpires” adjudicated losses based on their professional

judgment and ensured that players followed the movement and combat rules.

Combat modeling gained increased sophistication in 1914 when English mathematician Frederick Lanchester [16][36] proposed two sets of differential equations as the basis for determining the outcome of a battle. The equations sought to relate force ratios (attacker: defender) to loss rates. The first set of equations is called Lanchester's Linear Law, or sometimes his "unaimed fire" equations. They are:

$$\frac{dA}{dt} = -nAD$$
$$\frac{dD}{dt} = -mAD$$

where A is Attacker Strength, D is Defender Strength, t is time and n and m are rate coefficients developed from subjective evaluations of relative merits of equipment and tactics used by each side.

These equations relate the change of the attacker and defender strength in any time unit to a fixed proportion of the aggregate number of troops on both sides. They fail to account for positive feedbacks (such as increased aggressiveness caused by perceived success) or negative feedbacks (such as fear or disorientation caused by being surprised by an enemy). They also disregard the effects of terrain or unit movement during battle [19]. For instance, if one unit spreads its forces across a wider front, it presents a more difficult target for an enemy to fire at, so the proportion of soldiers killed in a time step should decrease. This battlefield adjustment is not explicit in Lanchester's equations.

The second set of equations are called Lanchester's Square Law, or sometimes his "aimed fire" equations. Rather than losses being a proportion of the aggregate number of forces, losses to each force are a proportion of the number of troops firing at that force. The equations are

$$\begin{aligned}\frac{dA}{dt} &= -kD \\ \frac{dD}{dt} &= -cA\end{aligned}$$

where A and D are attacker and defender strengths and t is time as before, with k and c being rate coefficients that are a function of the probability of the firing force hitting its target. As in Lanchester's Linear Law, there is no adjustment for dispersion of forces across the battlefield. Nor are there benefits for movement, surprise, training or discipline. Presumably, some attributes, such as training, discipline, and relative weapon effects, were intended to be captured by n , m , k and c , but that assumes that these factors are somewhat uniform across both armies and in some way quantifiable.

Modifications of Lanchester's Equations, however flawed, as well as the venerable 3:1 rule, are at the heart of most combat models in use today. The accepted models remain linear and attritional in their approach. Although some advantages are given for attacking an enemy's flanks or rear (usually in the form of a scalar increase in weapon effects), forces generally do not surprise an enemy and drive it from the field. In most simulations, forces fight to a predetermined threshold without regard to the way combat actually develops. Also, forces fight at the maximum level of capability based on the characteristics of their weapons, without regard to the intangible issues such as training, command, or fatigue.

The problem of this approach is that combat is treated as a set of fixed, linear equations to be solved by Gaussian elimination. Combat is reduced from being the free-flowing result of the millions of interactions between soldiers, the terrain and machines to a Newtonian system where every action results in a predictable, deterministic reaction. But combat is not a grandfather clock, it is dynamic. The agents not only react to their environment, but their reactions adjust over time to the changing situation. Later in this chapter, the state of the art for combat simulations will be compared against the requirement to be dynamic.

2.1.2 Simplification by Exclusion

The other issue identified for force development is the combinatorial problem. To combat this the approach is generally simplification through exclusion of what is deemed the “less interesting” solutions. The alternative solutions are heuristically limited to a number that can be evaluated with available resources while still satisfactorily searching the universe of alternatives. An example of this process is when the M1 “Abrams” tank began development; the Study Director identified 128 potential capability combinations, which were whittled down to 72 potential candidates. That number was, in turn, reduced to a smaller number for actual analysis through a Delphi Process.[45] A problem with this technique was that the heuristics used by the participants depended greatly on their combat experience which, even when extensive in terms of time in combat, was generally narrow measured across the spectrum of conflict. The veterans of WWII, for instance, had years of experience in combat against a conventional, armored force, whereas Korean War veterans had experience against a conventional, infantry force. The Viet Nam veterans had experience against

an unconventional, infantry-based force. Few had combat experience across the range of conflict. The WWII veterans insisted the tank be designed primarily for anti-tank warfare. The Viet Nam veterans insisted on emphasizing the anti-personnel capabilities. The tank loader was given a machine gun simply because the Study Director felt it was a good idea. Although there still is not a better system to reduce the number of potential candidates to a manageable number, this method strikes the author as distinctly unscientific at best and potentially damaging to the force at worst.

2.2 Current Combat Models

A review of the current models in use in the Department of Defense for ground combat follows. A description of the major models is presented then reviewed for adequacy against the requirements outlined in Chapter 1.

2.2.1 Janus

The most widely used combat development simulation used by the US Army is Janus. It is a two-sided, real-time, “man-in-the-loop,” interactive model that uses players to command individual units. JANUS is completely free-play. Human players can adopt any tactics and form any plan. Players set paths for their vehicles to follow while the game progresses, attacking and defending against an enemy controlled by other players. The players react to situations by applying their military judgment in order to replicate actual combat. Neither side can see forces on the other side, except where intelligence assets have detected the enemy

or where units are in physical contact, in order to realistically portray the commanders' picture of the battlefield and elicit realistic reactions.

A "man-out-of-the-loop" approach has been attempted in order to get the high volume of runs required to draw statistically valid conclusions about alternatives. The concept is to play a scenario until all participants agree it is representative of how a battle should unfold given the relative capabilities and tactics of the two sides. The computer then iterates the scenario to approximate the mean for selected measures of effectiveness. The approach does not take into account that the combat action will evolve differently if conditions change. For instance, if a unit is unlucky and draws multiple successive low (although random) numbers and is wiped out early, the plans of the other units, and perhaps the overall force, should change to adjust for the unit's loss. In this use of JANUS, units continue on the paths and timelines designated in the initial run, regardless of the altered situation.

In both the man-in-the-loop and man-out-of-the-loop techniques, changes in the number or type of equipment requires changes of tactics that must be implemented by subject matter experts controlling the forces. The man-hours required for each changed situation precludes most studies from including more than a handful of alternatives.

Janus also suffers from the effects of a learning curve. Players naturally become better at playing the game as they perform runs. As a result, later runs generally receive higher scores and are not directly comparable with those from earlier runs. Although much time is spent designing the experiment to minimize these problems, they can never be eliminated.

2.2.2 Modular Semi-Automated Forces (MODSAF)

Semi-Automated Forces (SAF) are computer-generated forces (CGF) that can respond to specific battlefield occurrences [62]. The US Army, in conjunction with the other services, is pursuing the MODSAF program as a training device that allows military units to train while minimizing the overhead required to portray adjacent and enemy units. The concept is that friendly and enemy SAF can operate on the simulated battlefield with manned forces so realistically that the players cannot distinguish manned forces from SAF. SAF can be used as friendly forces beside the manned force on a flank or as enemy forces tailored to look, act and perform as a chosen enemy army. Manned units use vehicle simulators to replicate their combat vehicles and allow them to see and hear the adjacent forces as they move in concert across simulated terrain.

SAF can be configured to behave in very realistic, but limited, ways. For instance if a leader orders a unit to cross a bridge, the SAF vehicles line up and cross without the leader having to micro-manage each vehicle. A parameter database contains behaviors required of each SAF based on its capabilities and role. Rote actions, such as those to be taken upon chance contact or actions required when under artillery attack, are automatically followed by SAF. SAF do not make battle plans or change their tactics based on changes in the equipment. The SAF commander must make those changes during initialization of the wargame or dynamically, during play, as the situation develops.

MODSAF has been developed as a training device, and therefore is focused on providing a realistic training environment. It runs in real-time, in order to allow training units to become familiar with how long tasks take and practice synchronization of all elements of

combat power.

2.2.3 Shortcomings

Both of these models, although useful in many ways, exhibit shortcomings. The inability to automatically tailor the behavior of each agent to the situation in Janus results in the exploration of a severely restricted universe of alternatives when new equipment or tactics are proposed. Typically, a new combat system, such as a tank, replaces the old version in a simulation and the effects are noted. Scenarios are artificially limited by the capabilities of the analysts to play and analyze them. This approach is unsatisfactory for many reasons. Firstly, the new capability may require tactics to change. All efforts are made to determine the best new tactics required using subject matter experts, but the time and manpower required for even simple changes can be prohibitive. If the change to equipment or organization is novel, there may even be additional runs required to determine the best tactics.

A new capability in one type of equipment may dictate a change in the number or quality of other pieces of equipment. The combinatorial complexity caused by evaluating every capability and quantity of each type of equipment that could bear on the situation prevents this from being explored except in the crudest terms. Recalling the combinatorial complexity of the three combat system example used before indicates that Janus is inappropriate to meaningfully explore this solution space since the man-hours alone would be prohibitive.

Even if an organization had the manpower and computing power to conduct an exhaustive search of the solution space, enumerating the solutions is not straightforward. There

are often both positive and negative interactions between capabilities. Determining the net result is problematic. For instance, adding armor increases survivability against a hit. The additional weight of the armor, however, negatively impacts the speed of the vehicle making it an easier target to hit. The added weight also impacts fuel consumption potentially causing a change of unit capabilities and, by extension, its employment. Better vision and weapon range improves survivability by allowing units to disperse over a larger area, but requires more fuel to move increased distances and probably requires faster vehicles, increasing the fuel requirement further. Assessing the net effects of all these changes and determining the proper tactics to account for them is difficult and increases the modeling time required.

MODSAF is designed for training, not combat development. As such, it replicates combatants well enough that soldiers undergoing training cannot tell machine-assisted forces from manned forces. It does not, however, run autonomously or change its plans based on a changing situation without human intervention. It cannot be run much more quickly than real-time since it relies on human players. This limits its ability to produce the runs required to search large solution spaces and develop statistically significant data in a reasonable time. It is well suited for what it was designed to do, but it is not a combat development tool.

Combat presents a mixed-integer, non-linear stochastic optimization problem where the interactions between agents are dynamic and must be dictated at run-time. Current models are linear. Their plans are static without the intervention of human players, forcing the options to be artificially limited. Chapter 3 outlines some methods being proposed in simulation optimization to mitigate these problems.

Chapter 3

Modeling Combat as a Complex Adaptive System

The Lord said to Gideon, “You have too many men for me to deliver Midian into their hands”... So twenty-two thousand men left, while ten thousand remained. But the Lord said... “There are still too many men. Take them down to the water, and I will sift them for you there”... The Lord said to Gideon, “With the[se] three hundred men... I will save you and give the Midianites into your hands. Let all the other men go, each to his own place.”

–Judges 7

It is enticing to rush headlong into modeling combat as a Complex Adaptive System given the apparent applicability and quantity of interest. Ilachinski is credited with the initial research into modeling combat as a Complex Adaptive System [34]. His modeling environment, EINSTEIN, has been used in some theoretical studies to explore its applicability to ground combat. Epstein, *et al*, used an agent-based model to explore civil disobedience

situations and the effects of policing and military action [20]. Many, like Czerwinski, have theorized that we cannot understand the current form of combat without thinking about it as a complex adaptive system [12]. Lesser, *et al*, Erlenbuch and Woodaman theorize that the current conflict scenarios that the United States faces, terrorism and low-intensity conflict, must be thought of as Complex Adaptive Systems and that combat must be modeled as such for the modeling to be appropriate [55][21][92]. Kewley has succeeded developing tactical-level orders by modeling small-unit combat as a Complex Adaptive System [46]. Goble has theorized the applicability of modeling combat as an alternative to current linear approaches [27]. Gill, *et al*, in New Zealand and Australia, have used agent-based distillations (as they call them) to study the human dynamics of combat [57][24]. It is important to note that, despite the interest from many researchers, little formal analysis exists to show that combat indeed fits the definition of a Complex Adaptive System. This is, no doubt, in no small part due to the lack of an agreed definition for a Complex Adaptive System itself. This section will examine the many definitions and synthesize them into one definition. It will then compare combat to that definition to show the applicability of the approach. Finally, it will explain the agent-based model developed to conduct the research.

3.1 Definition of a Complex Adaptive System

Despite the widespread interest shown in many disparate fields, there is little agreement on the precise definition for Complex Adaptive Systems. Researchers, it appears, have defined Complex Adaptive Systems in ways that fits their research goals and methods, mixing concepts from other fields in with nomenclature of their own. This is not a criticism; defining

Complex Adaptive Systems in the taxonomy of the field of research allows ready application in the field, but it is important to understand how these systems are viewed across the relevant research in order to apply the concept to the problem under investigation. Without establishing a framework, the assumption that combat can be modeled as a Complex Adaptive System hangs unsupported in the air.

Holland [31] posited seven characteristics of Complex Adaptive Systems that have been accepted as the “gold standard” in some form by most researchers. These seven characteristics are:

- **Aggregation.** Aggregation is used in two senses: First, that models are made up of smaller components aggregated into the larger model. Second, and more important, aggregation is the emergence of large-scale behaviors from the combined interactions of individual agents. This behavior may not be predictable, but can be explained after the fact.
- **Tagging.** Tags are attributes of agents recognizable by other agents. These might be size, shape or activity of ants that sends signals to other ants.
- **Non-linearity.** The relationship between system inputs and outputs is not definable by a set ratio. Holland’s example is the lynx-hare populations captured over time by the Hudson Bay Company. The populations oscillated between times of feast or famine in a distinctly non-linear manner.
- **Resource Flows.** Flows refer to the transmission of information, energy, or goods across a network. This is most evident in economic models, but has analogues in other systems.

- **Diversity.** Agents differentiate as they adapt to fill specific niches in the system. Removal of an agent from a system will result in a number of adaptations where the remaining agents seek to assume the role (or at least gain the resources) of the missing agent.
- **Internal Models.** Agents operate on the basis of local knowledge, which drives a set of assumptions about the general state of the system in order to make decisions. The quality of these models is directly related to the long-term viability of the agent. If an agent is “wrong” enough about the state of its system it will cease to exist.
- **Building Blocks.** Internal models rely on a limited sampling of the constantly changing environment, but models can only be useful if situations are repeated or the models will become inappropriate, and, therefore, inadequate for continuation of the system. The component agents find themselves facing similar situations but perhaps in different sequences making their experience continually novel but, nonetheless, their models remain appropriate.

Other researchers, such as Voss [88], have reduced this number to just 5 requirements. He agrees with Holland that Complex Adaptive Systems require Internal Models, Building Blocks, and Emergence (like Holland’s Aggregation or Kauffman’s self-organization [42]). He further posits that Complex Adaptive Systems have the ability to exhibit novel behavior when subjected to a changed environment. Voss’ novel behavior requires systems to adapt to meet the new challenge. Voss further proposes that Complex Adaptive Systems require the presence of multiple agents exhibiting both diversity and complexity. Systems with a

small number of agents would become trivial to analyze and would limit the adaptability of the system.

Steels, in his work on the nature of intelligence, focused on four attributes of systems: self-maintenance, adaptivity, information preservation and spontaneous increase in complexity [82]. Steels' self-maintenance refers to the property that these systems actively establish and rebuild themselves by drawing materials from the environment. This has also been called autopoiesis [58][59]. Adaptivity, as for Holland and Voss, indicates an ability to change structure or function in the face of environmental opportunities. Preservation of information allows the system to be independent of the existence of individual agents. This allows the elimination of agents without detriment to the existence of the system.

Spontaneous increase in complexity refers to the property that Complex Adaptive Systems will develop an increasing number of parts, the interrelationships between these parts will become more complex, behaviors will become more complex or parts of the system will combine to operate as a component part of a higher-level, more complex system.

Dooley has distilled his definition to three behaviors and a description of the underlying agents [14]. In his characterization of Complex Adaptive Systems, order is emergent rather than predetermined, consistent with Holland. He further states that the system's history is irreversible and the future is often unpredictable. The agents operating in (or making up) the system operate independently with schemata that determine how they view the world and how they react to what they perceive.

Others have tried a different approach, making the definition simpler, and, as a result, much broader. Bankes, working at Rand [7], has proposed the characterization of

Complex Adaptive Systems such that “no model less complex than the system itself can accurately predict in detail how the system will behave at future times.” Ilachinsky [33] has proposed that they are “non-linear, dynamical systems composed of many interacting semi-autonomous and hierarchically organized agents continuously adapting to a changing environment.”.

If we compare these definitions to the real-world systems that spawned the research of Complex Adaptive Systems, economies [3], ecologies [87], biologies [66], webs of corruption [79], as well as the human brain [82], among others, we can see value in all these definitions. Although each confirms Gell-Mann’s observation that scientists would rather share each other’s toothbrush before sharing their nomenclatures [23], each accepts implicitly or explicitly that Complex Adaptive Systems share the following characteristics. They:

- Consist of multiple interacting agents where agents are defined as independent acting entities that have attributes and operate on an internal model, or rule set, that governs their actions and reactions.
- Adapt at the atomic (agent) and/or the system level. Adaptation can be changes in the attributes of the agent or system or the rule set that they operate under. As such, they develop novel responses to changing inputs.
- Self-organize and, as a result, achieve stability without external input. The stable states are explainable, but not necessarily predictable.
- Exhibit interesting and complex behavior which implies non-linear, if not chaotic, behavior.

This definition captures the essence of the definitions above and passes the acid test, which is to take commonly accepted Complex Adaptive Systems and compare them to the definition. Without belaboring each point, economic systems, biological and ecological systems all seem to fit this definition. As a result, this will be used for the remainder of this work.

3.2 Applicability

Based on this composite definition, the next step is to determine if combat can be said to be a Complex Adaptive System. Each of the four parts of the definition will be addressed in turn. Should combat fail any of these parts, it will be judged to not fit the definition of a Complex Adaptive System, and a different modeling approach adopted.

3.2.1 Multiple Agents

Armies consist of large numbers of agents, from tank crews to artillery crews, infantrymen to truck drivers. Each fits the description of agents in that they are independent, have physical attributes, and act on internal models. That is, each tank crew and each infantryman must perceive the situation and make a decision how (or even) to follow his orders because he is often out of sight of the commander that gave them. As Major General Robert Scales has noted, when the Captain of a ship orders a turn, everyone aboard turns. In ground combat, each soldier is a freethinking actor that relies on his discipline, training, camaraderie, intelligence, knowledge of the situation and courage to turn with his unit.

These things he relies on are the internal models, or rule-sets, that he uses to sample his environment and determine his actions.

3.2.2 Adaptation

Armies, and the agents that constitute them, adapt as their experience grows. Soldiers adding armor protection to their vehicles to protect against experienced threats in Iraq is but one example. The Department of Defense is currently undergoing a massive transformation program to adapt to the threat from terrorism and so called “small wars.”¹

Confronting new enemies, as when the Coalition in Iraqi Freedom shifted from fighting the tanks of the Iraqi Republican Guard to fighting irregular forces, causes changes in tactics from the individual- to Army-level. Physical changes, such as changing vehicle types and organizations, are structural adaptation. Changes in tactics are adaptations of the internal models that agents or groups of agents (units) use. Clearly, combat fits this part of the definition.

3.2.3 Self-organization

Self-organization, or emergence of stability, is apparent in combat. Forces flow and collide in ways that belie the individual nature of the agents. It is commonplace for observers and historians to describe the movement of armies as “waves” or as to attribute to them anthropomorphic descriptions such as “brave,” “determined” or “ragged.” Despite the chaos

¹To differentiate what is going on in Iraq today from, say, war with the old Soviet Union.

often associated with direct combat, units form and disaggregate to accomplish missions and as a result of action. Those that show more cohesion, that is, those that more often achieve stability, are generally more successful. Armies achieve this stability without external input and can then be said to be self-organizing.

3.2.4 Non-linearity

As shown in the first chapter, non-linearity is an intrinsic property in combat and the reason that current modeling approaches are unhelpful. The examples of small forces causing disproportionate effects are the norm rather than the exception, the explanation of which has eluded historians and computer scientists alike.

As combat fits all four portions of the definition of a Complex Adaptive System, the premise that it is, and can be modeled as one, is accepted and a model can be developed accordingly.

3.3 Model Description

The computer model has been developed using the methods recommended by Parks and Leemis [65] in six phases.

- Description of the problem to be modeled in general terms.
- Development of a conceptual model of the problem.
- Conversion of a specification model based on the conceptual model.

- Development of a computational model.
- Verification that the computational model is in keeping with the specification model.
- Validation that the computational model is consistent with the conceptual model.

3.3.1 General

There is any number of aspects of combat that can be used to develop a model. This research is focused on developing combat systems and is a proof of concept, so the problem can be simplified to include two types of combat systems placed in a specific scenario.

The model will simulate the interaction between tanks and artillery. These systems perform distinct battlefield functions. Tanks find enemy forces and engage them by direct fire with their own weapons or call for indirect fire from other systems. Artillery pieces position themselves out of range of enemy direct fire to provide that indirect fire. Tanks move together with other tanks, balancing the weight of fire available from massed tanks with the susceptibility that massing provides to enemy fire.

A realistic scenario that can be adopted is a battalion-level attack against a company-sized force in a prepared defense.² The friendly, or Blue, battalion must capture an objective where the Red company is located. Red is tasked to defend that objective. This scenario has been chosen to focus the research on a manageable number of systems. As shown

²This initially places a set of 50 tanks and 18 artillery pieces attacking against 15 defending tanks supported by 6 artillery systems. These numbers and the types of systems will change as the solution space is searched, but this serves as an illustration of a starting point, or base case.

earlier, the dimensionality of even a small number of systems will cause the solution space to become very large. This provides a sufficiently large solution space to search and yield some useful results while providing a bounded problem for this concept exploration.

3.3.2 Conceptual Model

As discussed earlier, when equipment capabilities change, the rules by which the equipment is employed must change as well, so each system will exhibit physical as well as behavioral attributes. Physical attributes will range from types of weapons to the power of sights. Armor protection is a key defense mechanism, so many levels of protection will be available. The addition or subtraction of armor affects the performance of other aspects of each system, so speed and fuel requirements vary accordingly.

The forces are placed appropriately for an attack and a defense. Attacking forces are positioned, initially, out of physical contact with the defender so as to allow them to approach the objective and encounter the enemy. Placing them in immediate contact would bias the results towards systems that acquire and fire at enemies quickly without allowing any benefits from maneuver or the complementary use of multiple types of systems.

As forces come into contact, the sight system on board allows acquisition of the enemy at a range consistent with its capabilities. The acquiring system then calls for indirect fire and engages with direct fire within its capabilities.

Movement of the systems depends on their physical relationship to their fellows, the enemy and the objective. Units approach their objective while they perceive a reasonable probability of success and gain confidence from the presence (and nearness) of their fellow

systems. The three priorities, movement to the objective, movement with friendlies and attaining a proper position to attack (or avoid altogether) an enemy, constantly tug at each system. The priorities change as the perceived threat to the system changes.

Based on the perception of the situation, forces can continue their attack or defense, or break off the engagement. The action ends when attacker is in possession of the objective or the defender has successfully defeated the attacker.

3.3.3 Specification Model

The specification model requires determination of the structures and the states that will be modeled as well as the criteria for the state changes. In this model, there are two types of systems. Each set of system types, that is each combination of a type of tank and a type of artillery system, has a large number of potential states, which correspond to the number of solutions available in the solution space. The type of equipment that makes up the system governs the physical attributes of each system. The tactical attributes are made up of the relative importance of the three movement priorities at the perceived threat. These physical and tactical states will be developed in the following sections.

3.3.3.1 Physical States

Only a portion of the many possible characteristics of each system need be available for selection, because some characteristics are derivative of other physical attributes. Speed, for instance, a key system attribute, is a result of interaction between protection (the weight of the armor) and engine capability. Survivability is a combination of speed, protection, target

acquisition and weapon capability. As a result, the following list of attributes constitutes the set of physical states for each system.

- Weapon Type
- Ammunition Type
- Engine
- Amount of armor protection
- Target acquisition system
- Ammunition capacity

For tanks, the possible weapon types are: missiles, smaller, faster-firing guns, and larger, slower-firing guns. Ammunition is selected from missiles and conventional gun ammunition. Five potential engines are available. Armor can vary between very thin (0m) and very thick (1.5m) at 0.1m increments.

Each solution selects one of four target acquisition systems: direct view optical, infrared, thermal, and millimeter wave radar. The ammunition capacity will be allowed to vary between 20 and 70 rounds.³

For the artillery piece, the same set of capabilities is available. The weapon and ammunition types are tailored to reflect the purpose of artillery on the battlefield, but the options are similar in number. The engine, armor, and target acquisition systems select from choices similar to those available for the tank. Ammunition capacity will cover a larger range, from

³A round is defined as one missile or one bullet fired from a gun.

20 rounds to 100, to reflect the higher ammunition usage of artillery compared to current tanks.

3.3.3.2 Tactical States

The tactical states are a combination of the movement attributes and the perceived threat.

The movement rules are:

- Maintain formation with other friendly forces.⁴
- Move into an advantageous position in relation to the enemy.
- Move to the objective.

The priority of each rule will depend upon the agent's mission and its perceived threat. The threat perception is influenced only by what the agent can see, and can vary from system to system based on their location relative to other friendly systems and the enemy.

The mission and threat environments are partitioned into nine combat levels. Each level has a tuple that gives the relative priority of each of the three movement rules. For example, as the threat increases, the priority for staying in formation could increase, decreasing the priority of moving to the objective. The tuples will be different for tanks and for artillery in order to allow the tactics of a tank and an artillery piece to be replicated accurately. The Mission/Threat levels are:

- No threat.

⁴The appropriate distance between vehicles was coded in the chromosome and allowed to vary from 25m to 200m.

- Attack, Low Threat.
- Attack, Medium Threat.
- Attack, High Threat.
- Attack, Panic.
- Defend, Low Threat.
- Defend, Medium Threat.
- Defend, High Threat.
- Defend, Panic.

A sample table of tuples is shown below:

Table 3.1: Typical Rule Set Tuples.

	Rule 1	Rule 2	Rule 3
Mission/Threat	Friendly	Enemy	Objective
No Threat	10	0	1
Atk, Lo	2	5	1
Atk, Med	3	10	1
Atk, Hi	1	10	0
Atk, Panic	0	10	0
Def, Lo	1	0	2
Def, Med	3	0	2
Def, Hi	1	1	1
Def, Panic	0	1	0

In this example, the nine Mission/Threat levels are shown with the relative weight of the three movement rules. In the *No Threat* combat state, it is very important that the

tank get into formation with other friendly vehicles, so the value for “Rule 1: Friendly” in the tuple is 10. There is no weight assigned to moving to a good location versus the enemy since there is no threat. Moving to the objective rates a priority of 1.

Similarly, in the *Atk, Lo* state, maintaining formation is important, receiving a 2, but not as important as moving against the enemy, which rates a 5. Moving to the objective rates only a small value; in this case a 1.

These values are relative values, so a (1,1,1) is equivalent to (2,2,2). We can also say that in *Atk, Lo* it is 40% as important to stay in formation as it is to attack the enemy, but in *Atk, Med* it is 30% as important.

3.3.4 Computational Model

The set of states defined in the specification model was turned into a computational model by determining the simulation method, laying out the movement methods, and implementation of the physical and tactical attributes available to each system.

The model is an agent-based model using Mobile Autonomous Agents (MAA)⁵ moving on a two-dimensional lattice. The agent types are defined as tank and artillery systems. Next-event simulation was used where events are scheduled discretely, as their movement and actions dictate, in a global list. The update for each agent is asynchronous. Events are either moves or shots. As an event occurs, the agent surveys its environment, determines the next event, and schedules it. “Collisions” between agents are avoided by preventing

⁵A full explanation of an MAA is contained in section 4.3.

agents from occupying the same spot. If two agents select and are scheduled to move to the same location, the earlier moving agent occupies it. The later-moving agent will divert to a nearby location when it is his time to move.

The tank and artillery attributes were coded as genes in a chromosome. This convention was adopted based on the work outlined in the next chapter, which allows searching the solution space with a genetic algorithm.

There is no explicit commander on either side in the simulation. Like ant simulations, tanks and artillery communicate through passive stigmergy—that is, through their actions and interaction with the environment. The tactics embedded in each agent result in emergent overall action.

Each system scans the visible region by comparing its own location with the location of other agents by scanning linked lists of friendly and enemy forces. If the distance is smaller than the visual range of the system, the friendly or enemy is counted. This vision approach was adopted over having each agent scanning its entire visual area for enemy to speed the simulation without loss of fidelity. The number of agents will remain relatively small while the size of the visual area increases as a square of the range. Scanning a visual area requires $O(r^2)$ time where r is the visual range. This could bog the simulation down when visual range becomes large, whereas the small number of agents (initially less than 100) can be scanned in linear time relative to the number of agents.

The operational area is a 20,000-meter by 20,000-meter area represented by an 800x800 grid. Each lattice point is 25m away from its von Neumann neighbors.⁶ Movement is allowed in the x and y planes. Changing the apparent height of the vehicles as they move and become set in position simulates terrain. This “pseudo-terrain” allows full freedom of movement for all agents, but gives credit to agents that stop to fire (or are defending) for using all available local cover.

Threat is defined two ways, the force ratio of enemy to friendly agents and the distance to the nearest enemy. If no enemy agents are in sight, the agent is in the *No Threat* state. If the force ratio⁷ is below 0.3 and no enemy is within $\frac{2}{3}$ of the enemy’s weapon range, the state is “Attack, Low Threat.”

As the threat level rises, either by the force ratio increasing or by an enemy coming close to the system, agents will enter a defensive combat state. When the force ratios improve either through destruction of or retreat by the enemy or through increase in the number of friendly agents in contact, the agents will adjust their Mission/Threat levels and return to the attack.

Movement is performed like Craig Reynolds’ Boids [73]. Agents prioritize the movement rules and select a move based on the most eminent rule. For instance, in Rule 1 (maintain formation) the agent attempts to move to a spot at a 45 deg angle from its two nearest neighbors at a distance specified in the characteristics of the agent. Tanks normally move in a wedge formation to present the maximum number of weapons toward an enemy while

⁶The von Neumann neighborhood contains the four positions at the cardinal directions from the lattice point. The eight adjacent positions constitute a Moore neighborhood.

⁷Computed as number of enemy agents divided by number of friendly agents.

presenting the smallest target. In the simulation, the agent finds the best location relative to the nearest two friendly agents and computes the distance to that location. The distance multiplied by the Rule 1 priority value from the tactical tuple results in a priority value for moving to that location.

The simulation then determines the priority of occupying a location relative to the enemy. If any enemy is in sight, it determines the best firing location against the visible enemy. The spot cannot be outside the range of the agent's weapons (or he could not fire at the enemy) but should not be so close that the enemy has a high probability of killing the agent. The agent finds a location that best fits the criteria, computes the distance, and multiplies the distance by the Rule 2 element from the tactical tuple to determine the priority value for moving to engage the enemy.

In the same way, the tank computes the distance to the objective. The distance is multiplied by the Rule 3 element of the tactical tuple to determine the priority of moving to the objective. The highest priority value is the most eminent rule and the agent moves in accordance with it. To smooth the movement of the agents and prevent agents jumping to a distant location in a single event, each agent is limited to moving just one square in its Moore neighborhood. The agent moves one grid toward the location dictated by the most eminent rule. At each grid, the agent reevaluates the threat and its neighbors, then determines and schedules the next move.

Both the Attack and Defend missions have a *Panic* state to allow agents in a high threat environment to panic with a small probability. This is to allow forces to attempt to break off the attack or defense, and possibly abandon the mission. This was done to

introduce the effect discussed in Chapter 1 where real soldiers, either through loss of nerve or misunderstanding of their environment, react inappropriately.

If enemy agents are in range, a tank will shoot at the highest-threat enemy agent and call artillery on the others. Tanks can shoot from stationary positions or on the move. If they shoot on the move, a small accuracy penalty is assessed on the shot, however. Artillery must be stationary in order to fire. Each event is scheduled and executed in turn. As each event is complete, the vehicle determines and schedules the next event.

Probability of hit, P_h , is determined by the range and accuracy of the firing weapon and the size of the target. Friendly and enemy movement degrades accuracy. A draw from a uniform distribution compared to the P_h determines if the target is hit. Probability of kill given a hit, $P_{k|h}$, is determined from the amount of armor on the target and the attack angle of the shot. Armored vehicles have their armor concentrated in the front 60° to protect the crew from the majority of hits. $P_{k|h}$ decreases if firing at the front of a vehicle. A draw from a uniform distribution compared to the $P_{k|h}$ determines if the agent is killed. If so, it is removed from the lattice.

When a vehicle is hit, but not destroyed, its capabilities are decreased by an arbitrary value drawn from a uniform distribution.⁸ When the cumulative effect reaches 1, the vehicle is destroyed and removed from the lattice.

The Red force agents will replicate forces using Soviet-designed equipment. Although the Soviet Union is not in existence, many states around the world use its equipment and,

⁸No data is available on the most likely amount of damage a tank can expect if not killed, so this is a simple way to assess a penalty for being hit. If more data becomes available, this penalty assessment can be adjusted.

as a group, constitute a reasonable threat set. This also allows the system threat data to be derived from open sources such as Jane's Defence Publications.

The simulation stops when one side is defeated, defined as when Blue reaches the objective, is destroyed or runs out of time. These criteria allow Blue to receive credit for mission accomplishment by either destroying Red or driving Red from the objective, but require mission accomplishment in a reasonable amount of time. This prevents giving credit solely to attritional solutions, one of the critiques of current wargames. It also ensures that Blue cannot simply avoid contact with the enemy and still receive a score.

The fitness function will require mission accomplishment (seizure of the objective) as the first test before any fitness score is given. If a solution does not accomplish the mission, its score will be 0. That is because the first test in combat is mission accomplishment. This is analogous to retaining only solutions in the feasible region of a simplex search. Once the mission accomplishment gate is passed, fitness will be scored with the following function:

$$\frac{200b}{b_0} + T - t$$

where b and r are the numbers of Blue and Red agents at the end of the simulation and b_0 and r_0 the respective numbers at the start. T is the arbitrary cut-off time for the simulation and t is the amount of time required to accomplish the mission.

This scoring system gives a solution two points for each percent of the friendly force that survives and one point for each minute under an arbitrary time limit that blue accomplished the mission. The scoring system is designed to reward accomplishing the mission as quickly

as possible while protecting friendly forces. It uses percentages of Blue agents that survive in order to allow comparison between solutions with differing numbers of vehicles.

This is intentionally a simple scoring system. More elaborate scoring systems could be devised to take into account many other attributes, but the purpose of this dissertation is to demonstrate a proof-of-principle that this modeling and optimization approach is a viable alternative to the linear, attritional approaches available.

The full code for the computational model is not reproduced here, but is at Appendix A of this dissertation.

3.4 Verification

To ensure that the computational model conformed to the specification model, extensive test were performed to measure each agent's movement priorities and its ability to shift from one mission/threat state to another. The movement rule priorities were evaluated off-line for a small number of agents and compared to those generated by the model.

When the agents successfully passed the movement tests, they were allowed into direct and indirect fire contact where they had to not only move, but also evaluate their threat levels and adjust their movement rules accordingly. Once successful, they were allowed to generate direct fire shots and calls for indirect fire.

These shots were evaluated to ensure that agents were hit in the probability expected, and that the effects of the shots were accurately recorded. If the agent was hit, but not

killed, the appropriate hit penalty was evaluated and compared to that produced in the simulation. If killed, removal from the simulation was confirmed.

When the “mechanical” workings of the model were deemed to be satisfactory, it was compared to the conceptual model in the final phase of model development.

3.5 Validation

Validation that the model accurately portrays the conceptual model is the most difficult phase, given the emergent behavior of a Complex Adaptive System. Verification of individual actions does not ensure that they will combine to replicate emergent behavior exhibited in practice. Validation, then, has to take a top-down approach where the overall results are compared to known situations to determine if the actions are reasonable and explainable.

To do this, a visual output module was added to the model to allow researchers to watch the interactions of the agents and the flow of the simulation. Initial values for the Blue and Red agents based on current systems⁹ were used to “calibrate” the initial runs. With some trial and error for the tactical rules, the runs were recognizable as typical combat formations and movement. Results from many runs were evaluated in order to ensure that the observed results were typical. When the results were deemed satisfactory, the physical and tactical attributes were varied to ensure that explainable results were developed in multiple situations. Only after extensive testing was the simulation deemed sufficient and runs made with solutions selected by the genetic algorithm.

⁹With data supplied by the Federation of American Scientists web site and Jane’s Defence Systems.

Chapter 4

Searching the Space

It is an error to imagine that evolution signifies a constant tendency to increased perfection. That process undoubtedly involves a constant remodeling of the organism in adaptation to new conditions; but it depends on the nature of those conditions whether the directions of the modifications effected shall be upward or downward.

–Thomas H. Huxley

After addressing the issue of dynamic modeling in the last chapter, the next issue to be addressed in this dissertation is searching the hyperdimensional solution space. First, the nature of the solution space needs to be determined, and then a solution developed. In this section, the fitness space will be explored and a framework for categorizing fitness landscapes discussed. Once the state of the landscape is known, then existing search methods will be discussed with one being adapted to use in this problem.

4.1 Characterization of the Fitness Space

Many search techniques have been advanced to solve problems with high dimension hyperplanes and very large solution spaces. Absent in this discussion has been a standardized representation of the landscapes themselves, although many authors have indicated that the shape of the landscape is the most important factor in determining problem “hardness” given a particular solution technique. The literature refers to landscapes as “lumpy,” “rough,” “noisy,” “deceptive” and even “porcupine-quilled,” [2] [68] but only general definitions of these terms has been provided and even less has been discussed on how the landscapes came to be categorized as such. Simple landscapes, that is, landscapes with small dimensionality, can be graphed and categorized “by eye” but interesting landscapes are, by their nature, high dimensional and resist simple visual categorization. It appears that this issue has been ignored for two reasons. First, the landscape metaphor, introduced by genetic biologist Sewall Wright in 1932, is so strong that researchers in all fields grasp it as a concept immediately without further study [37]. Second, exploration of the underlying landscape has been foregone in favor of exploration of the solution methods themselves. These reasons avoid the key factor affecting suitability of the selected solution method.

The shape of the fitness landscape results from two factors, the problem itself and its abstraction for solution. The relationship between those two parts describes the resulting shape of the landscape. An example can be $y = x^2$ which, when using real numbers, is a smooth, continuous function [48]. If it is abstracted using binary notation and sorted by

Hamming distance, it will provide a non-smooth landscape¹.

Landscapes that result from binary representations are very common when using genetic algorithms as a solution method, which makes it all the more surprising that there has been no common way to characterize those landscapes. As the problem under study is a mixed-integer, non-linear problem whose solution is represented by a binary string, this discussion will limit itself to that class of problems.

Clearly the entire landscape cannot be mapped to determine the shape because of the size and because the high dimensionality restricts visual mapping. Selected portions can be mapped if the size and dimensionality are reduced sufficiently. To do this requires selecting a point, defining its neighborhood, and providing a standard process by which the neighborhood is to be represented. The point can be selected at random, and, if done a sufficient number of times, these multiple looks can allow a general characterization of the overall landscape.

Defining the neighborhood is not as simple as selecting the starting point. One proposal has been to map all the possible vertices that can be reached in one crossover function from two solutions [49]. This is intuitively attractive, for two reasons. It limits the size of the partial landscape to $2C(2, n - 1)$ where n is the number of genes in a single solution chromosome. The size remains manageable even for very large landscapes. The representation of the neighborhood, however, is problematic. In what order do the solutions appear on

¹Kingdon and Decker used this example to show that landscapes were a result of the interaction between the problem representation and the solution method. I propose here that many solution techniques could be attempted given the problem and its abstraction, so the solution is not as important to the shape of the landscape as the abstraction of the problem.

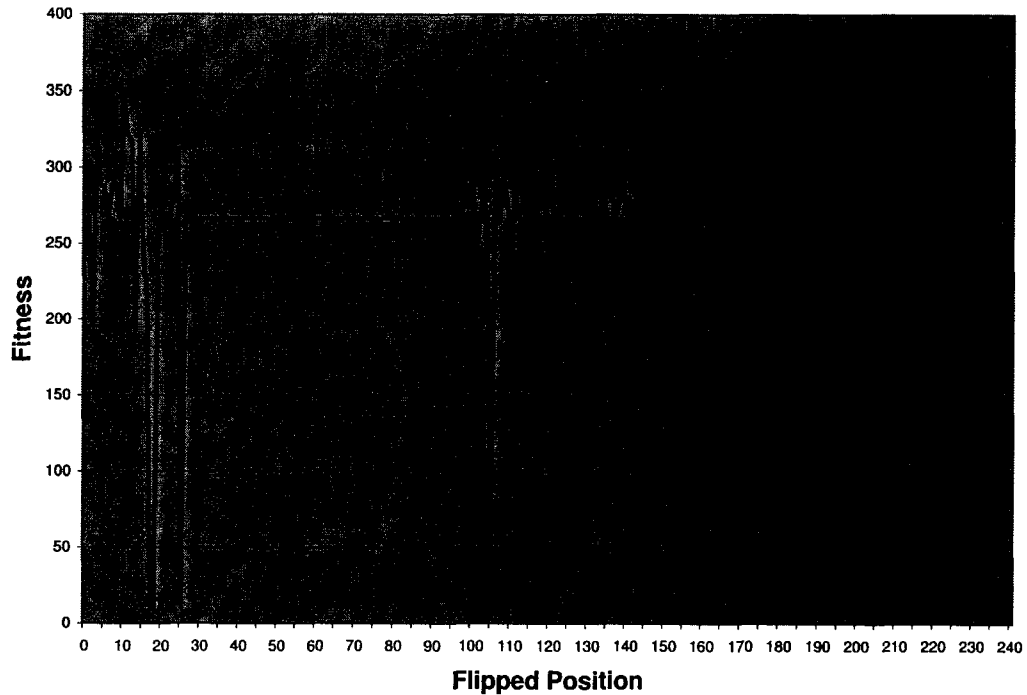
the x and y axes of the graph? The representational method greatly affects the perceived shape of the landscape.

This dissertation proposes a different approach, defining the neighborhood as all points that have a Hamming distance of either one or two from the randomly selected point. This limits the size of the partial landscape to $n - 1$ and $C(2, n - 1)$ respectively. It recommends an obvious two or three-dimensional presentation of the resulting partial landscape. The position of the genes “flipped” from the original solution can define the position on the x or x and y -axes. The greatest benefit of this system is that it has wide applicability across all landscapes that use binary strings.

Two examples follow that using this method to restrict the size and dimensionality of the subject function. In the first, all solutions with a Hamming distance of one from a randomly selected solution are evaluated and the resulting partial fitness landscape graphed. The x -axis represents the position in the binary string that has been changed from the original solution. The y -axis represents the fitness of the solution. This graph indicates that around a given solution, the fitness landscape is generally flat, but is punctuated by spikes both of both improved and degraded fitness. This can be done at several randomly selected points across the landscape in order to gain an understanding of the shape of the landscape.

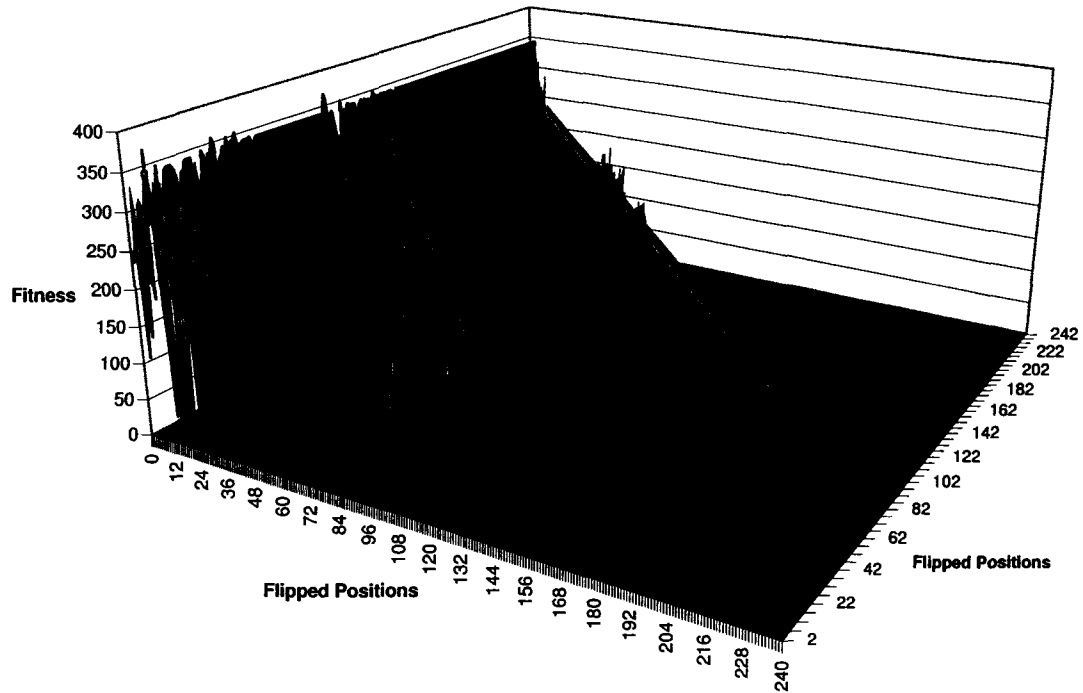
The following figures present the neighborhood of all points with a Hamming distance of two away from a randomly chosen solution. This three-dimensional half-matrix provides a wider view of the points around a solution and a more complete characterization of the neighborhood. This graph shows that the landscape around this point is again flat with

Figure 4.1: Two-dimensional Landscape Representation.



distinct regions of increasing and decreasing fitness.

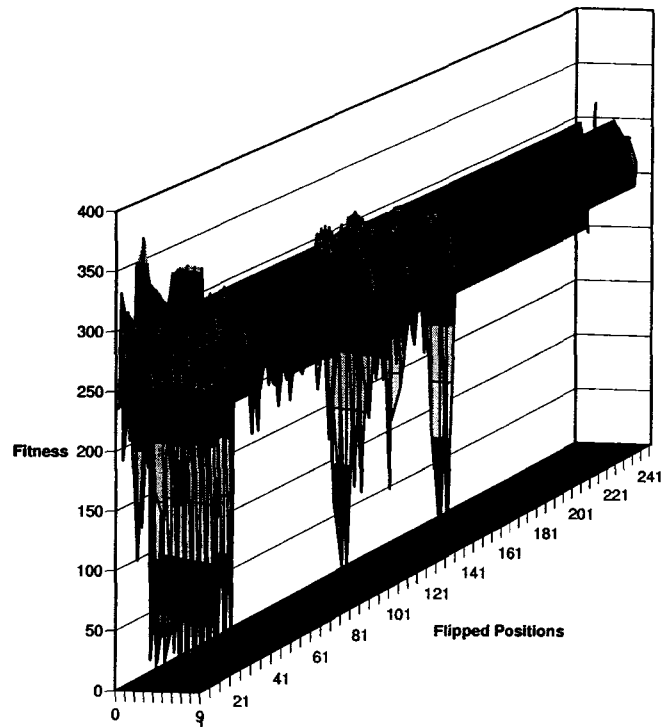
In order to focus on a sample of the partial landscape under study, the area corresponding to flipping the binary value contained in the first 10 positions has been extracted. This allows more detailed analysis of the areas that are sensitive to changes in the position values. In this case, that analysis reveals that fitness is highly sensitive to the values contained in the first 25 positions with a region of sensitivity between the 105th and 139th positions. Although the actual results of the modeling will be covered in Chapter 5, the sensitivity exhibited in the first 25 positions indicates that fitness is sensitive to the physical capabilities of the tank and tactics used when threat is low. The sensitivity between the 105th and 139th

Figure 4.2: Three-dimensional Landscape Representation.

positions indicates that changing the capabilities of the artillery piece can impact the fitness of a solution more often negatively than positively. The broad areas between those regions indicates that the solutions are relatively insensitive to changes to the values contained there.

This information is valuable for two reasons. First, some solution techniques are inappropriate to particular landscapes. Knowledge of the landscape shape allows a researcher to select a more amenable solution technique or problem representation. Second, this information could indicate areas of sensitivity and insensitivity to changes to particular genes in the chromosome. Identifying areas of improvement could alert a researcher to ensure

Figure 4.3: Close-up of a Portion of the Three-dimensional Landscape Representation.



that those landscape regions are specifically searched. Identifying “sleeper” genes, or areas of insensitivity, could allow a shorter solution chromosome and a correspondingly reduced solution set, improving overall search performance.

4.2 Available Approaches

4.2.1 Derivative Approaches

Simulation optimization requires a method to search through the hyper-dimensional solution space presented by the options and a way to evaluate each solution dynamically. Several

methods of searching the solution space are discussed below followed by an approach to building dynamic models.

4.2.2 Frequency Domain Method

Simulations in which each attribute set requires a simulation run are called “run-oriented” simulations [77]. An alternative to run-oriented simulations is to vary the inputs in a known manner during the simulation run and evaluate the effects on the simulation. The Frequency Domain Method is used to test the sensitivity of model output to input parameter changes. This method is appropriate to modeling a system like a power generation plant that operates at a steady state. The output of the plant can be monitored as the attributes change and conclusions drawn as to how to optimize the system. Combat operations are not steady state; they unfold over time as forces move and present themselves to the enemy. A sort of steady state could occur if forces become stalemated. In this case, however, each force will attempt to find a solution to break the stalemate and move away from steady state. Combat is best represented by terminating simulations [54] which makes the frequency domain method inappropriate to evaluate most combat operations.

4.2.3 Differential Qualitative Analysis

Differential Qualitative Analysis (DQA) is a method that perturbs an attribute of a system then follows the perturbations through a system in a forward-chaining method to determine the overall effect [89]. This works well if the interactions can be quantified and if they occur in the same order. In combat, small changes in the early portion of a battle can result in

later interactions occurring in different orders, or not occurring at all. This would make DQA inappropriate to apply to combat simulations.

4.2.4 Response Surface Methodology

Response Surface Methodology (RSM) attempts to cut off large portions of the solution space by evaluating a portion of it and determining the direction of maximum improvement [53][6]. An ensemble of model runs is made to determine an initial response surface constructed of many single or multiple linear regression models. The gradient representing the direction of greatest ascent ² is determined. More simulation runs are made to determine another response surface in the direction of the gradient and the process iterated until a solution is found. This method works well in an objective landscape that presents wide slopes leading to a single global maximum. Complex functions that result in a landscape characterized by sharp ridges and multiple local maxima are unsuited for this kind of analysis. A system with a large number of attributes which interact is likely to present just such a complex landscape with many local maxima [42].

Grier, *et al*, attempted to use RSM to find the best mix of Air Force aircraft and weapons in a scenario set in Southwest Asia against an Iraqi-based threat [28]. They used 26 runs of a model called THUNDER³ and captured 34 output metrics. These metrics were mapped to seven meta-variables, which corresponded to seven of the nine campaign objectives identified in the experimental design. In the end, only five of the seven meta-variables could be fitted

²If maximizing the objective function.

³THUNDER is a deterministic air combat simulation that uses fractional exchange rates (meaning that losses can be in fractions of an aircraft or target). It takes a long time to set up and run each iteration. This long set-up and run-time is the motivation for an approach like this that limits the runs required.

with a response surface, and of those, four had a correlation coefficient below 0.9, indicating that the fitness of the surface was poor. Their approach made a valiant attempt to overcome the shortcomings of the model, but does not offer a way ahead. RSM appears to be of limited use in optimization of combat simulations.

4.2.5 Evolutionary (Derivative Free) Approaches

Evolutionary Algorithms include Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms, and Genetic Programming [13]. Other evolutionary methods have been suggested, but remain at their core, modifications of these broad approaches. The basis of evolutionary algorithms is rooted in nature where plants and animals compete and cooperate in search of resources [60]. The more successful become stronger and are more likely to mate, passing along their genes to their offspring. The offspring in turn, compete and if successful, pass along their genes. In most species, two members combine their chromosomes in sexual reproduction. This increases the diversity of the off-spring [42] and allows for new and unique combinations of the attributes of the species to be “tested.” The successful combinations repeat the process, the unsuccessful combinations die out. Mutations occur in the combination process introducing new, unique combinations of attributes, which serves to push the search into novel parts of the solution space and serves to prevent the search from converging prematurely on a local maximum.

4.2.6 Evolution Strategies

Evolution Strategies (ES) were developed in the 1960's in by I. Rechenberg [60]. Real-valued attributes of a solution are each represented as genes. The genes are the building blocks of a double string, called a chromosome that represents a candidate solution. The first gene in the double string is the value of the point in the search space for that attribute. The second gene is the standard deviation allowed for the value of that gene. Once the objective value of the solution (the parent) is determined, the genes are mutated by an amount drawn from a distribution dictated by the standard deviation to produce another solution. The offspring is then evaluated. If it improves on the parent, the parent is discarded and replaced by the offspring; if not, the offspring is discarded. The process repeats until the candidate solutions stop improving.

This approach has shown promise in engineering problems such as designing airfoils and other continuous optimization problems, but has not been used in a mixed integer, linear or non-linear, simulation optimization. There does not appear to be a reason why the approach cannot be modified to include integer-valued or binary attribute values, but there is no current research where this approach has been used.

4.2.7 Evolutionary Programming

Lawrence Fogel, apparently independent of ES, also developed evolutionary programming in the 1960's. It seeks to predict changes in the environment based on the previous and current states of that environment [13]. Each solution is represented by a Finite State Machine (FSM), which examines a string of symbols (each relating to states of the environment

ordered over time), and seeks to predict the next symbol. Solutions are ranked on their predictive ability; the higher ranked solutions produce offspring based on random mutation of the states of the machine, either changing a state or adding to the state string. The offspring are evaluated and rank ordered with the parents. The higher-ranking solutions remain in the pool and the lower ranking solutions are discarded. The process iterates until the solutions stop improving.

This remains an active area of research and may hold some promise for combat simulations [11]. Efforts are being made to use this to develop combat plans.

4.2.8 Genetic Algorithms

John Holland developed genetic algorithms in the 1960's to study how natural adaptation might be replicated by computer systems. In genetic algorithms, as in ES, the attributes of an agent are represented as a string of genes, but in a single chromosome. In Holland's genetic algorithm, the alleles (values of the genes) were binary. Since then, work in the genetic algorithm field has expanded to include integer and real values [60] but most success has been with binary alleles. In a genetic algorithm, an initial generation of chromosomes (solutions) are developed either through design by the experimenter or at random. The chromosomes are evaluated for fitness and chosen to survive or reproduce based on their relative fitness. In this way the better solutions are more likely to either survive or to reproduce, passing along good genes to an offspring. Mating occurs through crossover where chromosomes from two parents are "broken" at a (usually) random location. The partial chromosomes are recombined with the complementary chromosome fragment from the other

parent to form two unique offspring. Solutions that are not chosen to survive or to mate are discarded, keeping the population sample constant size. These surviving chromosomes and the offspring are subsequently evaluated. The selection/evaluation process iterates until the solutions stop improving.

Genetic algorithms make the greatest use of mutation. Each gene has a non-zero probability of “flipping” outside of the crossover function. This, as discussed earlier, prevents premature convergence by forcing the search off a local maximum. Much research has been done to determine the appropriate mutation rate. Anastasoff has even researched allowing the mutation rate to evolve, without, however, improvement in performance [2]. Although no “correct” rate has been identified, the consensus is that a stationary rate of .001 is generally appropriate.

Genetic algorithms have shown great promise both in evolving both structures [51][87] [5] and rule sets [74][35]. Previous works have focused on the development of either structures or rule sets, but research on evolving both simultaneously is beginning to emerge [78]. This appears to be a viable approach to addressing the combat modeling shortcomings outlined in previous chapters.

4.2.9 Genetic Programming

Genetic programming involves using computer programs as agents that perform a task and replicate themselves either through combination with other programs or through self-reproduction [70][85]. Results thus far include developing an artificial ecology of computational entities existing in a virtual environment. The entities vie for computer time with

the faster and smaller programs being judged more “fit.” Interesting mutations have occurred which allowed programs to discard their own ability to copy themselves in favor of using the ability of another agent to reproduce. This method has shown promise in finding efficient algorithms optimized to a task [69], but does not appear to be useful for simulation optimization.

4.3 Artificial Life

4.3.1 Definition

ALife was first proposed by Chris Langton in the late 1980’s and is a non-traditional discrete optimization technique that uses evolving agents [91]. This dissertation has referred to agents in previous chapters without providing a full explanation of what they are. Agents in this context are simulated objects that interact with their environment using an internal rule set. The agents can either modify the environment of all other agents simply by reacting to the environment as they find it. Agents often evolve over the course of the simulation through learning, reproduction with other agents, self-replication or directed replication through means of an evolutionary algorithm.

Agents can communicate with other agents directly through token passing or indirectly through modification of the environment. A simulation of decision-making that uses as its agents a team leader and the team members can incorporate direct communication between agents [40]. A simulation of ant colonies relies on pheromone trails laid down by ants as they move through the environment to communicate the presence of food [9].

Agents make their own decisions based on the environment (which includes the other agents) without outside direction. In this way, local rules can result in an emergent, global behavior, not explicitly built into the simulation.

Since AL is a relatively young field, numerous names are used for very similar concepts but it can be said to consist of two distinct branches, Cellular Automata (CA) and Mobile Autonomous Agents (MAA). CA consist of agents statically arrayed in a lattice. Each agent interacts with the agents in its immediate neighborhood⁴. MAA move across the lattice to interact with other agents and accomplish goals. Because of the dynamic nature of MAA, it has been used to evolve structures and rule-sets for mobile agents.

The biological analogy of these simulations has spawned a number of successful attempts to model natural systems such as birds [17][86][22], ants [9], termites and turtle populations [72]. Man-made systems such as traffic have also been studied by using MAA [63]. These descriptive models have been useful for understanding the dynamics of the natural world, but do not use the evolutionary nature of agent-based models. Other MAA have been used to evolve physical structures and rule sets, which are more applicable to modeling combat.

The biological analogy also changes the optimization taxonomy. Nature has found “fit” solutions for the given environment. Accordingly, the objective function in this kind of simulation is called the fitness function since it measures degree of fitness. Fitness is not the same as optimality. No one can doubt that Neanderthal man was sub-optimal in terms of intelligence, but he was capable of adapting to his environment and was “good enough”

⁴This neighborhood could be a von Neumann neighborhood consisting of the points at the four cardinal directions or could be the Moore neighborhood that includes all eight surrounding grid points.

to allow him to survive and reproduce for some 100,000 years. His design was the “fit” solution necessary in his environment.

4.3.2 Evolving Structures

MAA have been used to evolve structures in two ways. The first is a discrete competition to perform a task such as moving across an environment [51]. In this approach, agents are paired in competition, with the winner allowed to reproduce with other winners. The winner and its offspring then compete in the next generation of agents. Competition continues until agent capability stops improving.

The other method is to allow each of the agents to inhabit an environment and compete for resources [87][18][69]. The agents move to attain a goal such as acquiring resources. Those that perform better, becoming stronger and living longer, have more opportunity to mate and reproduce. This perpetuates the stronger attributes through natural selection while the weaker attribute sets die out.

4.3.3 Evolving Rules

Rule sets have also been evolved using MAA. In these cases, the physical aspects of the agents have been kept static, but the rules by which they move and accomplish goals have changed. Examples have been soccer playing simulations [84], combat simulations [39][47], and the game of tag [74].

In these simulations, a small number of agents are simulated and a fitness value for their performance determined. A genetic algorithm then searches through the solution space of rules until a most-fit solution is found.

4.3.4 Evolving Both Structures and Rules

There is little current research in evolving both structures and rules at the same time. Sims has, however, evolved agents that capture a goal in one-on-one competition [78]. He randomly generated agents and paired them against each other to capture a cube placed at the center of an arena. The least fit agents were discarded to make room for the new offspring. The most fit agents reproduced “sexually⁵” and the population paired off in another tournament. The fitness function was simply the time required to capture the cube and to carry it back to the agent’s own starting location.⁶ Sims found that novel structures and rules evolved where some agents attempted to “protect” the cube to prevent the opponent from capturing it while others relied on speed to snatch the cube and return before the opponent could react. This indicates that there is no inherent limitation to searching the solution space of both physical attributes and rules simultaneously.

4.4 Ant Colony Simulations

A specific branch of Mobile Autonomous Agent research is Ant Colony Simulations. These developed from studies of ants and an attempt to model their living and colonization habits.

⁵Meaning use of a crossover and mutation function.

⁶Less time is better.

These simulations have proven so rich that they have been expanded from simply describing ant behavior to become an on-going area of optimization research. Ant agents act based on their current state without memory of previous events [90]. As such, they can be simulated by Finite State Machines (FSMs) in order to solve problems such as shortest path between two points, shortest path between a subset of points in a network, and shortest Hamilton cycles in a network (Traveling Salesman Problem, TSP).

Actual ants exhibit the so-called “coordination paradox” [83] in which they do not communicate directly, as with tokens or physical language, but yet coordinate to build and maintain nests and forage efficiently for food. They seem to communicate indirectly using stigmergy [8]. Stigmergy is the reaction to changes in the environment either actively caused by the ant or as a side effect of its actions. For instance, a real ant lays down a trail of pheromones when returning to the nest with food. The presence of pheromones indicates to others that food is available along the trail. Ants react to that change in the environment by following the trail, finding the food, and laying their own pheromone trail behind them as they return. If no food is available at the food site, the ants return, but do not lay a pheromone trail. The trail evaporates over time and disappears so that ants do not continue to visit an empty food site. This kind of stigmergy is called *active stigmergy* [32].

Ants also react to environmental changes not directly caused by other ants using passive stigmergy. For instance as corpses build up in the nest, ants consolidate them with other corpses. These corpse piles then become large enough to trigger a reaction in the ants to carry all the corpses out of the nest [9]. No central direction is given for collection or disposal of corpses. Individual ants respond to the stimulus of the presence of corpses to

generate a collective housekeeping behavior.

This indicates that each agent can have an individual internal rule set but still act collectively. This has direct applicability to combat simulations where the activity of agents is governed by the decisions made by individual agents. If each agent has the overall framework of what should be accomplished based on a situation, the group of agents can perform collective tasks such as conducting an attack or a defense. This is a reasonable representation of real combat given that low-level leaders⁷ are trained to apply the proper tactics in each situation and to know how they fit into the tactics and mission of the overall unit. Each of these low-level leaders with their respective combat systems could be represented as an agent. Each agent could have the overall tactics embedded. The Army spends a great deal of time and resources to develop and train “battle-drills” where tank commanders and small unit leaders react appropriately in response to an overall situation with little or no communication. Experience at the NTC and in combat shows that these battle-drills are useful techniques at battalion level and higher.

Ant Colony simulations also lend themselves to using evolutionary algorithms to optimize their performance. White, *et al*, used a genetic algorithm to improve an ant simulation attempting to optimize path-finding [90]. When finding a point-to-point path, using a genetic algorithm decreased the time required to find the optimal path by 25%. When finding a path through a subset of points in a network, the time required was reduced by 26%. To find a minimum Hamiltonian path through a set of points, the time was decreased 24%.

⁷Including infantry squad leaders and tank commanders.

4.5 Enhancement Through Co-evolution

Empirical studies have shown benefits to co-evolving two or more types of agents (which can be thought of as species or tribes) in the same artificial world but the theoretical basis for that improvement has until now been elusive. The advantage derives from the fact that, as each species tries to climb its own fitness landscape, it deforms the environment for the other species. The second species reacts to the change, which deforms the landscape of the first. Each species prevents the other from being locked onto a local maximum. The two species continue this process until they achieve equilibrium at the most-fit co-solution.

Kauffman, et al, [43][41][44][42] studied this coevolutionary phenomenon and used it to help solid state physicists understand spin-glasses. Spin-glasses are a type of disordered ferro-magnetic material. The direction of each “spin” in relation to the others affects the overall energy of the spin-glass. Kauffman used this model to show the benefits of using co-evolution to find a fit combination of spin directions.

Suppose a landscape consists of N “spins”. The spins for this example are binary and result in some energy level between 0.0 and 1.0. The energy level for the landscape is the sum of the energy levels of the spins. Each spin is independent in that it can be changed individually, but is connected to K other spins with a resultant collective energy level. In genetics, this is referred to as epistatic coupling of genes, where the activation of one gene may cause activation or inactivation of others. This results in a complex energy fitness landscape where the energy contribution of each spin must be specified for each of the 2^{K+1} configurations that the spin, and the K spins that affect it, can be arrayed.

Kauffman defined the energy level as the average of the energy contributions of the spins and expressed it as follows:

$$E\{s\} = \frac{1}{N} \sum_{i=1}^N E_i^{(K)}(s_i : s_{i_1}, \dots, s_{i_K})$$

where $E\{s\}$ is the average energy level, $E_i^{(K)}$ is the energy level of spin i which is connected to K other spins.

In an attempt to optimize (in the case of a spin glass, lower) the energy level of the system, one could calculate the improvement derived by flipping each spin and choose the largest improvement. Unfortunately, this could lead to a local optimum where no one-flip neighbor improves the energy level but combinations of flips could, in fact, improve the solution. Finding these combinations would involve calculating all the one-flip, two-flip, up to N -flip changes—in other words, calculating the energy level for all combinations of spins, or evaluating 2^N solutions. If the problem could be solved with this brute force approach, there would be no need to use any sort of optimization technique.

Kauffman broke the lattice into P “patches” of size $p \cdot q$. The number of patches is equal to $\frac{N}{pq}$. He then examined one-flip improvements for each. Since spins were connected across the patch boundaries, lowering the energy level in one patch could increase the energy levels of other patches. Moving along an improving fitness landscape deformed the landscapes of the other patches. The energy level of the new problem is:

$$E\{s\} = \frac{1}{N} \sum_{P=1}^{N_P} \sum_{i \in P} E_i^{(K)}(s_i : s_{i_1}, \dots, s_{i_K})$$

where P represents the patches to be optimized and i represents the spins that are summed by patch.

Kauffman attempted three approaches. First, a spin was chosen at random and flipped if it improved the patch that contained it. Second, each one-flip change in a patch was evaluated and one that improved the energy level was randomly chosen. Last, all one-flip changes in a patch were evaluated and the best flip selected. The patches were chosen in order, with one selection of all patches called a “generation.”

His results showed that, when using any value of $K > 0$, after five generations the average energy level found was better than attempting to optimize the entire landscape. The difference became more pronounced as K , that is, the number of connected spins, increased. This indicates that in the presence of epistatic couplings, a better solution can be found more quickly by breaking into patches. Co-evolution resulted in more rapid movement toward the optimum.

Hillis, using a sorting algorithm, also showed this result [30]. A sorting network is an algorithm in which a sequence of comparisons and data exchanges is performed in a predetermined order. These networks have great practical importance in switching circuits and routing algorithms for interconnected networks.

To test this approach, Hillis used a network where the number of data elements to be sorted (n) was 16. He chose this value for n since the problem was well studied. The best network found contained 60 comparisons [50]. He evolved a solution by starting with a set of random networks and testing them against a set of test information strings to determine their fitness (fewer errors equaled increased fitness). The least fit networks were eliminated,

with the remainder reproducing using crossover and mutation. The new solutions were then tested for fitness and the process continued until the solutions quit improving. By this method, Hillis found that he could evolve sorting networks of 65 exchanges, close to the best-known solution, but not as good.

Hillis then allowed the test cases to evolve in parallel with the networks. Their fitness criteria measured how many errors they caused. The fittest test cases were allowed to reproduce through crossover and mutation to evolve better tests. These improved test cases exploited weaknesses in the evolved sorting networks, which deformed the evolved networks' fitness landscape. The resulting networks were 61 exchanges long, an improvement from 65 and almost equal to the best known. This indicates that there is benefit to co-evolution when the fitness landscapes are coupled. The solution may not be optimal (although optimality is not precluded), but a very good solution to a complex problem can be found through co-evolution fairly rapidly.

These examples would indicate that other complex systems, like evolving single weapon systems in isolation, even if searching the solution space with a genetic algorithm, could result in less-fit solutions than if the systems are co-evolved. It appears that systems should be evolved together to be able to complement and improve each other. This addresses one of the major shortfalls of current combat models.

4.6 Co-evolutionary Theory

The strong empirical evidence that co-evolution improves solutions has led researchers to accept the phenomenon and exploit it in areas as diverse as the theoretical applications above and concrete applications such as the diagnosis of Breast Cancer [66], but there have previously been only “naturalistic” explanations of why improvement occurs, without a theoretical basis for the improvement.⁸

The naturalistic explanation is that, as each species evolves, it deforms the solution space for the other species. The species evolve in a continuing competition where evolutionary tension pushes them to climb to the global optimum. Species locating local optima may find that the changes in the opposing species quickly make that position non-optimal. This give and take continues until a joint, globally optimal location is found. This has been called the “Red-Queen” hypothesis⁹ where predators and prey must continue to evolve to remain at parity [64]. This explanation makes great sense and probably explains what is going on inside the genetic algorithm as it evolves to a global solution.

A more rigorous explanation is that co-evolution both increases the exploitation of fit schemata and increases the ability of a genetic algorithm to explore new solutions. This is very much a “have your cake and eat it” situation, as most times exploitation of “good” solutions comes at the expense of exploration and vice versa. For example, if the probability of mutation or crossover increases in a genetic algorithm, the potential for fit solutions to be

⁸Improvement being defined as arriving at a more fit solution or arriving at an equal solution faster.

⁹Named for the Red Queen in Alice in Wonderland where everyone had to run in order to remain in place.

destroyed increases. This risk is balanced by the benefits that exploration of the landscape brings. Much research has centered on how to balance exploration against exploitation.

4.6.1 Increased Exploitation

Co-evolution can be shown to improve exploitation of fit schemata as a result of increasing the probability of schemata surviving the crossover process when compared to evolution. The increased search range can be shown by the increased step-size afforded by co-evolution.

The probability of a schema surviving the crossover function is related to its *defining length*, that is, the distance between its most-distant fixed “genes” as measured on the chromosome representation. The longer the defining length, the more potential “cut points” exist between the fixed positions. This increases the probability that the crossover operator will fall between the fixed positions, disrupting the schema. This is called a “representational bias” against long schemata. In coevolution, two crossover points are selected (one in each set of system genes that make up the entire chromosome). If the crossover points occur both inside of the defining length of the schema or outside of that length, no disruption occurs and the schema survives [38]. The probability of a 2^d -order schema (that is with two fixed positions) surviving typical (single-point) genetic algorithm crossover is:

$$p_{s_1} = 1 - \frac{l}{L}$$

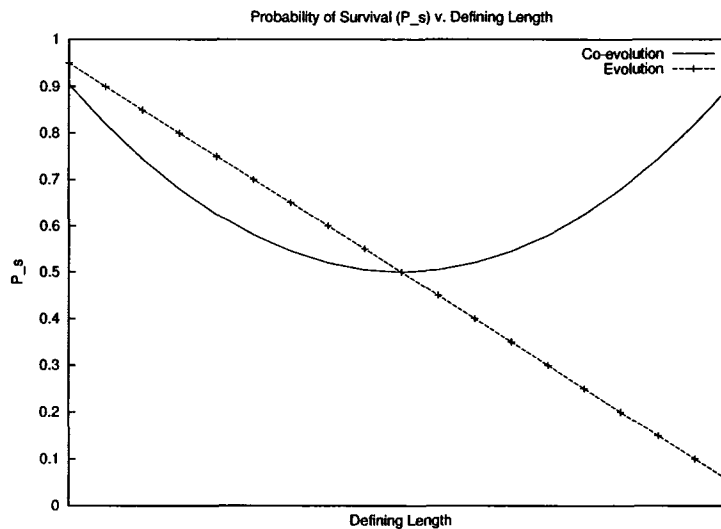
where p_{s_1} is the probability of the schema surviving a single cut, l is the defining length of the schema and L is the length of the chromosome.

For co-evolution the probability of survival of a schema is equal to the probability that both crossover points are either between or outside of the defining length or:

$$p_{s_2} = \left(\frac{L-l}{L}\right)^2 + \left(\frac{l}{L}\right)^2$$

where p_{s_2} is the probability of the schema surviving two cuts.

Comparing these probabilities, it is apparent that for very short defining lengths, $p_{s_1} > p_{s_2}$, but as the defining length approaches L , $p_{s_2} > p_{s_1}$. Graphically the difference is shown below and it is clear that the cumulative probability of survival of a schema under co-evolution is much higher than for standard genetic algorithm.



De Jong (et al) [38][81][80] called the area above the curves the “disruption area,” the area where a schema is disrupted. By integrating the equations for the two curves, the area below each line, the “area of stability” (or cumulative probability of survival for 2^d -order schemata) is shown to be larger for the coevolution case. The results are:

$$P_{s_1} = \int_0^L \left(1 - \frac{l}{L}\right) dl = \frac{L}{2}$$

$$P_{s_2} = \int_0^L \left[\left(\frac{l}{L}\right)^2 + \left(\frac{L-l}{L}\right)^2 \right] dl = \frac{2L}{3}$$

While this only accounts for 2^d -order schemata over the chromosome of length L , extending it to include higher-order schemata (those with more fixed positions) gives the recursive function:

$$p_s(L, l_1, \dots, l_{d-1}) = \sum_{i=0}^1 \left(\frac{l_1}{L}\right)^{2i} \left(\frac{L-l_1}{L}\right)^{2-2i} [p_2(l_1, l_2, \dots, l_{d-1})]^i$$

where p_s is the probability of survival of all ordered schemata, d is the total number of fixed positions, p_2 is the probability of survival of all schemata with a number of fixed positions from 2 to d , L is the length of the chromosome, l_d is the length of the schemata with d ordered positions.

To determine the cumulative probability of survival of a third order schema requires expansion of the equation and integrating twice, first as l_2 , or the shortest distance between fixed points, goes from 0 to l_1 and second as l_1 ranges from 0 to L giving:

$$P_s(L, l_1, l_2) = \int_0^L \int_0^{l_1} \sum_{i=0}^1 \left(\frac{l_1}{L}\right)^{2i} \left(\frac{L-l_1}{L}\right)^{2-2i} [p_2(l_1, l_2)]^i dl_2 dl_1$$

which expands to:

$$\int_0^L \int_0^{l_1} \left(\frac{L-l_1}{L}\right)^2 + \left(\frac{l_1}{L}\right)^2 \left[\left(\frac{l_1-l_2}{l_1}\right)^2 + \left(\frac{l_2}{l_1}\right)^2 \right] dl_2 dl_1$$

Performing the integration reveals that the cumulative probability of survival for 3^d -order schemata in coevolution is:

$$P_s(L, l_1, l_2) = \frac{L^2}{4}$$

This can be compared to the evolutionary case where survival of any schema is equal to the probability that the single cut falls outside the longest defining length, that is, the length between the most distant fixed positions. The probability equation is simpler to evaluate. Despite the higher number of defined positions it remains:

$$1 - \frac{l_1}{L}$$

Evaluating the area under the, now three-dimensional, curve requires a double integration as did the previous case, giving:

$$\int_0^L \int_0^{l_1} \left(1 - \frac{l_1}{L}\right) dl_2 dl_1 = \frac{L^2}{6}$$

Comparing this result to the survival of a 3^d -order schemata in coevolution, we see that probability of survival for a 3^d -order schemata is greater in coevolution than in evolution just as in the case of 2^d -order schemata.

Evaluating the probability of survival for schemata up to $d = 5$ where d is the number of fixed positions shows that this property holds for small values of d :

Table 4.1: Probability of Survival for d -order schemata

d	Coevolution	Evolution
2	$\frac{2L}{3}$	$\frac{L}{2}$
3	$\frac{L^2}{4}$	$\frac{L^2}{6}$
4	$\frac{L^3}{15}$	$\frac{L^3}{24}$
5	$\frac{L^4}{72}$	$\frac{L^4}{120}$

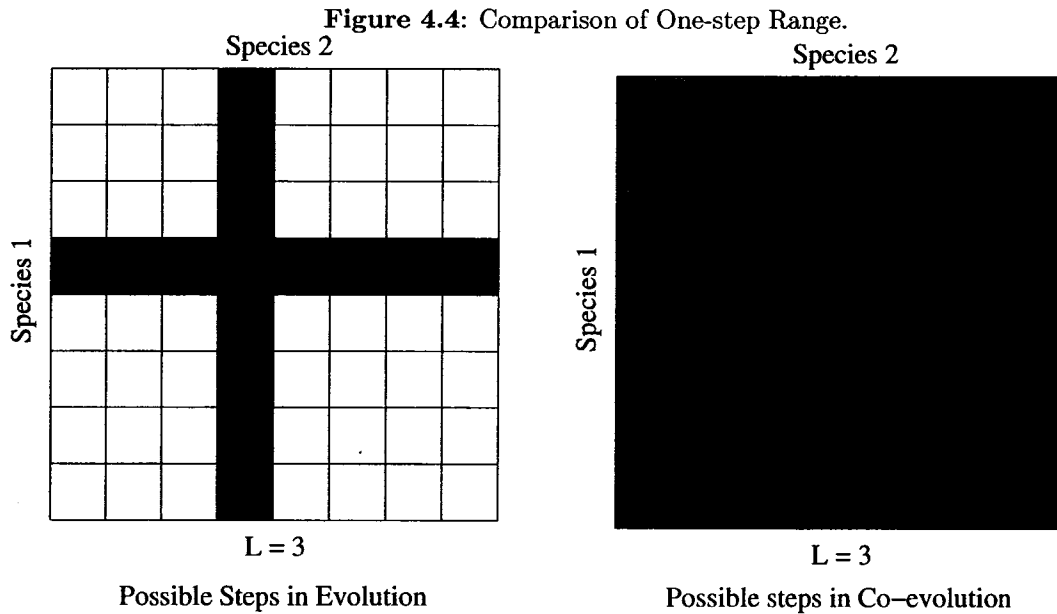
Using inductive reasoning we can prove that survivability of schema in co-evolution will always be greater than in evolution. We set our base case as $d = 2$. The area under the curve represents the probability of survivability. As we expand from a two-dimensional space to three dimensions ($d = 3$), the additional factors for co-evolution and evolution cases will be in the same proportions as with $d = 2$ making the volume described in the co-evolutionary case larger than the evolutionary case. Probability of survival, then, will be greater, as in the two-dimensional case. As the dimensionality increases, the proportions between the additional components remain, so probability of survivability for co-evolution will always be greater than evolution.

4.6.2 Increased Exploration

Exploration of schemata increases due to the larger step size available to each crossover action.¹⁰ In evolution, as a result of its reliance on single-point crossover, the number of possible offspring combinations one step from a set of parents is $2^{L+1} - 1$. In co-evolution, more solutions are available in a single step because the chromosome is cut twice. The number of possible offspring combinations available from each set of parents is 2^{2L} , which

¹⁰In this dissertation crossover is defined as the process where two chromosomes are cut at a selected point beyond which the “genetic material” is swapped to form two new chromosomes. In the case of two-point crossover, the genetic material between the cuts is swapped.

means that co-evolutionary available step-size will always be larger than that available in evolution. This is shown in the following graphs.



In these two graphs $L = 3$, where L is the chromosome length (in number of genes) for each of two species. The columns and rows represent possible solutions caused by the eight combinations of the three genes. On the left, the center-shaded block denotes a set of parent solutions. When a set of parent solutions is crossed using one-point crossover, the resulting offspring can be any solution denoted by the shaded area (including being mirrors of the parents). On the right, the same set of parents is mated using two-point crossover. The shaded area indicates the increased number of possible offspring combinations available. As can be seen, the entire universe of combinations is available in a single step, instead of the limited set available in evolutionary, one-point, crossover.

Increased step-size does not guarantee that fit solutions will be found faster than in a

simple evolutionary genetic algorithm. There may be situations where smaller steps are better. It does indicate, however, that the search will generally be more wide-ranging using co-evolution. Examination of more different regions of the fitness landscape in a fixed number of steps will tend to find better solutions and prevent the genetic algorithm from searching unproductive regions.

Improved retention of fit schemata, combined with the widened single step search distance indicates that, on balance, co-evolution improves the speed to get to a fit solution. Based on the theoretical improvement shown by a co-evolutionary genetic algorithm, this research adopted the following approach.

4.7 The Genetic Algorithm Approach

4.7.1 General Approach

A standard genetic algorithm was implemented to search the solution space for the most-fit mix of capabilities and rules. Although the systems were initially patterned on current equipment and tactics, they freely co-evolved. Based on the six physical attributes and the 27 tactical genes contained in the tuples (whose values ranged from 0-10), the tank and artillery piece each had about 10^{31} possible solutions.

The system chromosomes were linked to form a single chromosome with one end of the strand representing the tank and the other representing the artillery piece. The crossover point for each chromosome was drawn independently from a uniform distribution. Each end of the chromosome was split once and mated with the complementary portions of the other

parent's chromosome. Mutation was allowed at a rate of 0.001 per gene in keeping with contemporary research.¹¹ This is a reasonable starting mutation rate given the literature [56][60].

4.7.2 Selection of Solutions

Selection of potential parents was proportional based on the score generated by the simulation compared with the other solution scores in the generation. This gave more-fit solutions a higher probability of selection for crossover or inclusion in the next generation and penalized less-fit solutions.

Two conventions were implemented during the selection of the solutions for the next generation: elitism and increased probability of retaining unfit solutions. Elitism, that is, retention of the best solution in a generation, prevents the solution from being lost and the generational results from prematurely converging on a suboptimal solution [66]. Unfit solutions, that is, with scores less than 2.5% of the maximum attainable, were not eliminated out of hand. Instead, these solutions were arbitrarily awarded a small score to allow them to compete for retention and crossover in the next generation. This prevented unfit solutions that reside in otherwise fit areas of the solution space from being discarded reducing the ability of the co-evolutionary genetic algorithm to further explore that space. When the parent solutions for the next generation were selected based on relative fitness, this gave the unfit solutions a non-zero probability of surviving to the next generation. This did not decrease the ability of the algorithm to search for fit solutions. As Holland [61] has shown,

¹¹Mutation was allowed whether or not crossover occurred.

good solutions will still be selected exponentially for survival. It did, however, force the algorithm to search areas that might not get attention, thereby closing off potentially good solutions.

A cost constraint was instituted on the overall cost of the force to require the genetic algorithm to make trade-offs between system capabilities. Without such a constraint, each system will improve until reaching the solution with the largest number of the most expensive machines. As Emmeche [17] said:

“Evolution acts as a tinker who fixes a broken machine from materials at hand. Not every design is a good design, many are called but few survive. Instead of constructing few expensive complicated machines designed for few well-defined tasks, maybe a flock of small, cheap, perhaps rather unpredictable machines allowed to evolve naturally is better.”

Limiting the cost of the overall force, while allowing the cost of each system to be dictated by its capabilities, allows just the kind of trade-off that Emmeche discussed.

Chapter 5

Modeling Results

Man always fears the consequences of danger more than the danger itself.

–Maurice de Saxe

To test the theoretical findings in Chapters 3 and 4, the model and co-evolutionary genetic algorithm were implemented and the results compared with a standard (evolutionary) genetic algorithm. The results confirm the two theoretical expectations outlined in Chapter 4. First, the resultant solutions were generally better using co-evolutionary techniques. Second, co-evolution tested a wider range of solutions during its search. The results also confirmed the proposal in Chapter 3 that, since combat is a complex adaptive system, it could be modeled as such to get useful solutions. This is important because, although theoretical underpinnings for co-evolution as a search technique are good, unless the technique renders a useful solution, it is of little value.

Each of these findings will be explored, and the modeling results presented, in the following sections.

5.1 Improved Solutions

Search methods can be considered “better” if they either arrive at a more fit solution or arrive at an equally fit solution more quickly. In this research, both evolutionary and co-evolutionary genetic algorithms arrived at very fit solutions quickly, so overall fitness of the solutions was used as the standard.

To evaluate the evolutionary versus the co-evolutionary approach, a set of 30 solutions was randomly generated¹ for each run. This constituted the common starting solution-set for the two approaches. A simple genetic algorithm employing one-point crossover and a co-evolutionary genetic algorithm were run from that initial solution population until the respective algorithm quit making progress and converged on a solution. Crossover occurred at a rate of 0.6. Mutation occurred at a rate of 0.001/gene. In both approaches, elitism was implemented. All solutions competed for inclusion in, and to become parents to, the next generation.

As shown in the following table, after eight record runs,² the co-evolutionary approach resulted in more fit solutions in six of the eight. The fittest solutions found by run are shown in the table below:

The shape of the landscape can explain the relatively minor difference in fitness between the evolutionary and co-evolutionary solutions. As shown in Chapter 4, the landscape

¹Each binary gene was selected using random draw from a uniform distribution.

²More than eight runs were made but initial runs were used for model validation and not included in the production runs.

Table 5.1: Highest Fitness Found.

Run	Evolution	Co-Evolution
1	381.999	382.934
2	384.075	389.108
3	390.129	391.243
4	385.864	386.452
5	384.982	388.878
6	384.954	385.745
7	391.459	391.449
8	393.803	393.724

consists of relatively flat plateaus punctuated by isolated local maxima and minima. Both approaches were able to find higher (more-fit) plateaus, but co-evolution generally found more-fit solutions than did evolution. The fact that co-evolution was not the best in every case is a result of the stochasticity of the search technique. Any search technique has a non-zero probability of finding an equally or more-fit solution and the evolutionary technique did so in two of the eight runs, although the difference is very small in both cases. The co-evolutionary approach, however, was expected to result in better solutions and, in practice, did.

5.2 Increased Exploration

An advantage of the co-evolutionary approach was expected to be the exploration of more solutions as a result of the wider one-step search range available. To determine if this occurred in practice, the number of unique solutions visited was captured and the range from most- to least-fit solution in each generation was recorded. In both measures, co-evolution resulted in a broader search of the solution space.

In every case, more unique solutions were visited in the co-evolutionary case. This broader search increases the probability that the technique will locate a better solution. This is in keeping with the theoretical work from Chapter 4.

At the end of each run, generally after more than 30,000 generations, the co-evolutionary approach maintained a wider spread between most and least fit solutions. This indicates that, even after an extensive search and convergence on a fit solution, the co-evolutionary approach continued to include a wider range of solutions. The inclusion of an increased number of less fit solutions widened the search area, again improving its ability to find fit solutions long after the evolutionary approach converged on a solution. The results by run are shown in table 5.2.

An interesting side effect of maintaining a wider search longer was that the average fitness of each generation was lower in the co-evolutionary case, even though the most-fit score was generally higher. That is to be expected and is not a weakness of the co-evolutionary technique. It is something to be aware of when using a co-evolutionary technique.

Table 5.2: Objective Value Range of Solutions Found.

Run	Evolution	Co-Evolution
1	117.1	39.0
2	6.6	58.6
3	9.3	36.9
4	40.8	55.8
5	31.6	31.3
6	5.2	10.5
7	22.4	34.5
8	8.4	25.9

With the exception of the first and fifth runs, the results are completely in keeping with

the expected results. As in the search for more fit solutions, the stochastic nature of the approaches would preclude one approach always being better than another. Indeed, had the results been completely one-sided, there would have been concern that there were problems in the implementation. In this case, however, the preponderance of the runs bear out the theoretical expectations.

5.3 Appropriate Solutions

The final, and really most important, test for the ability of the approach of coupling a dynamic, agent-based simulation with a co-evolutionary genetic algorithm is that it resulted in militarily appropriate solutions. This is more a test of the ability of the agent-based model to adequately replicate combat, so only the fittest solutions found in the co-evolutionary runs were evaluated for relevance and military value. If serious flaws existed in the solutions, then there would be concern that the approach was flawed. The fittest solutions were also compared across the runs to identify similarities and differences. If the solutions are found to have a great deal in common, it is an indication that the approach found a common area of the solution space despite beginning at random locations; further justification that the approach is sound. If the solutions are explainable but widely different, it indicates that the approach is not appropriate for this fitness landscape, as defined by the problem and its abstraction. More work would need to be done to determine if the problem was the approach or the inappropriate landscape.

5.3.1 General

The results across the eight runs showed clear convergence on key system attributes. When offered choices between superior capabilities, with their attendant cost and size penalties or lesser capabilities without those penalties, the agent-based model replicated combat well enough to allow the co-evolutionary genetic algorithm to make reasonable, explainable choices.

System attributes that constituted most fit systems were clear for tanks. Due to the scoring system, which made speed a key factor, none of the most-fit set of solutions allowed the force to wait for artillery to move and have an effect on the enemy. As a result, no artillery capabilities were tested. In future work, the scoring system should be modified to place less emphasis on speed to conclude the mission and, by doing so, perhaps place a higher premium on combined action with the artillery. This could also have been an artifact of the single scenario with which the systems were tested. To provide conclusive evidence for system decisions, a number of scenarios should be evaluated in future work.

That being said, in this scenario, the solutions selected were explainable and appropriate, although not necessarily expected. Although each attribute will be discussed in detail, some general conclusions can be drawn. A single sight was selected across the most-fit solutions (that is, the top 11 solutions found across the eight runs).³ Medium armor protection was selected along with an inexpensive engine capable of moving a medium-weight vehicle. An inexpensive gun system was more often selected than more expensive (but more accurate

³The top solutions from each of the eight runs were compared. In addition, in three runs, very fit alternative solutions presented themselves. As the fitness differential was small, they were included to make a total of 11 solutions.

and lethal) missile systems. The tactics selected to complement these choices placed a premium on moving quickly to the objective until the enemy was found, then configuring the force appropriately relative to the enemy. Details of the attributes are listed below.

5.3.2 Sights

Four sights were available. In order of their relative cost, probability of detection and accuracy from lowest to highest they were: optical, infrared, thermal, and millimeter wave. In the 11 best solutions, the thermal sight was selected as the optimal balance between cost and capability. The salient capabilities and cost factors of each system are shown below.

Table 5.3: Sight System Capabilities and Cost

Sight	Prob of Detection	Cost(\$000s)	Accuracy (mils)	Range(km)
Daylight	0.4	10	0.8	2.0
Infrared	0.6	100	0.8	2.5
Thermal	0.8	250	0.5	5.0
Millimeter Wave	0.95	1000	0.1	10.0

The increased cost of the thermal sight over the optical and infrared sights was outweighed by the capability of the system. A more capable sight was available, using millimeter wave technology. The large cost increase for the millimeter wave sight, however, was not justified by its increased performance in this scenario.

5.3.3 Armor Protection

Sixteen levels of armor protection were available, from a thickness of 0m (no protection) to 1.5m of armor⁴ in increments of 0.1m. The more armor on a vehicle, the more survivable the system is against a hit by an enemy weapon. Increased weight brings penalties in weight, size and cost. More armor makes the vehicle heavier and slower, reducing mobility, or a more expensive engine. More armor also increases the dimensions slightly increasing the probability that it will be hit. It also makes the vehicle more expensive, reducing the number that can be purchased.

The co-evolving genetic algorithm selected armor protection from no armor to 1.3m thick, with a mean protection level of 0.71m. The Red Tank gun is rated at a penetration level of 0.55m, so clearly protecting against a penetration was a priority over weight and cost, but the overmatch between protection and the threat is not large. The thickness stated refers to the thickness of the frontal armor. Side and top armor is thinner, so a higher frontal armor value could also indicate that protection of other aspect angles warranted the increased weight and cost. Although one solution did select a high level of protection (1.3m), the high frequency of mid-level choices indicates a bias towards “just enough” protection.

5.3.4 Weapon Systems

Weapon systems showed the most variety. There were 16 weapon systems to choose from, 7 conventional guns, 6 guided missiles and 3 advanced technology electromagnetic guns. The

⁴As stated in earlier descriptions, this refers to rolled, homogenous armor; a standard gauge of protection.

conventional guns were cheaper, with a higher rate of fire, but less accurate. The missile systems were more expensive, slower to fire, but highly accurate. The electromagnetic guns had very high accuracy and penetration capability, but large system cost. Seven of the 11 solutions found used conventional guns, four selected missiles, and none selected the electromagnetic guns. The seven gun selections were confined to just three different options that balanced cost against capability. The selected guns were the midrange models available. The missiles selected were the low end cost systems that had adequate capabilities, indicating that increased capability was desired, but only when the increased performance warranted the increased cost.

Table 5.4: Selected Weapon Systems Capabilities and Cost

Type	Penetration (m)	Cost/Shot (\$000)	Accuracy (mils)	Cost/Wpn (\$000)
120mm Gun/M829A1	0.70	2.0	0.90	250
125mm Gun/BM42M	0.65	2.5	0.90	350
125mm Gun/BK29	0.55	2.0	1.0	350
AT-11 Missile	0.80	50.0	0.40	500
TOW-2B Missile	0.80	100.0	0.40	500

A review of the non-selected systems indicates that the cost of the missiles and the Electromagnetic Gun were not justified by their improved performance. Cost versus capability also appeared to be a factor in the guns selected.

Table 5.5: Capabilities and Cost for Non-selected Weapons

Type	Penetration (m)	Cost/Shot (\$000)	Accuracy (mils)	Cost/Wpn (\$000)
120mm Gun/M829	0.60	2.0	1.0	250
120mm Gun/M829A2	0.80	5.0	0.80	250
125mm Gun/BK27	0.60	2.9	1.0	350
140mm Gun	0.80	4.0	0.08	500
FOTT Missile	1.0	150.0	0.30	500
Javelin Missile	0.60	75.0	0.30	500
LOSAT Missile	2.0	250.0	0.08	500
Dragon Missile	0.20	40.0	0.30	500
Electromagnetic Gun	2.0	0.001	0.05	1250
Electromagnetic Gun	2.0	0.01	0.05	1250
Electromagnetic Gun	2.0	0.10	0.05	1250

5.3.5 Engines

Seven of the 11 solutions selected the cheapest, least powerful, engine able to move the vehicle with a medium level of armor protection (called the M60 engine here). One selected the LV100 engine. Two selected the M1 engine and just one selected the most expensive, hybrid engine. The reduced size of those engines saved some cost of armor protection (due to the smaller envelope that needed to be armored), offsetting some of the increased cost of the engines and decreased the vehicles probability of being hit (P_h). In general, however, solutions favoring the cheapest engine were dominant in these runs.

Table 5.6: Engine Cost and Parameters

Type	Cost(\$000)	Weight(metric tons)	Horsepower	Size(m^3)
M60	100.0	3.5	750	5.04
LV100	175.0	1.4	1500	1.99
AGT-1500	250.0	3.3	1500	3.95
Hybrid	500.0	1.0	900	2.00

5.3.6 Tactics

The selected weighting of the tactical rule sets for each of the solutions was slightly different, but some trends emerged across the 11 solutions. Generally there was a bias towards the rule dictating movement to the objective and some bias towards the rule ensuring proper alignment with respect to the enemy. Little value was placed on the rule that maintained formation with other friendly forces. Table 5.7 shows this more clearly. The columns represent the threat levels, the rows represent the predominant rule at each level of threat. The predominant rule shown could be a single rule, a combination of two rules, or balanced between all three rules. The number in each location represents the number of times that combination of rules was selected in the 11 solutions.

Table 5.7: Rule Predominance

Dominant Rule	No Threat	Low Threat	Med Threat	High Threat
Friendly	0	0	0	0
Enemy	1	3	5	2
Objective	3	3	3	3
Fr/Enemy	2	1	1	0
Fr/Obj	3	2	0	2
En/Obj	1	1	2	3
Balanced	1	1	0	1

When no enemy was detected, the 11 solutions selected solutions that, understandably, were weighted towards moving to the objective. Seven of the 11 solutions had a significant bias towards moving to the objective. When the enemy was detected, the enemy and the objective rules were valued about equally throughout the threat levels except at a medium

threat when there appears to be a bias towards the rule governing position relative to the enemy.

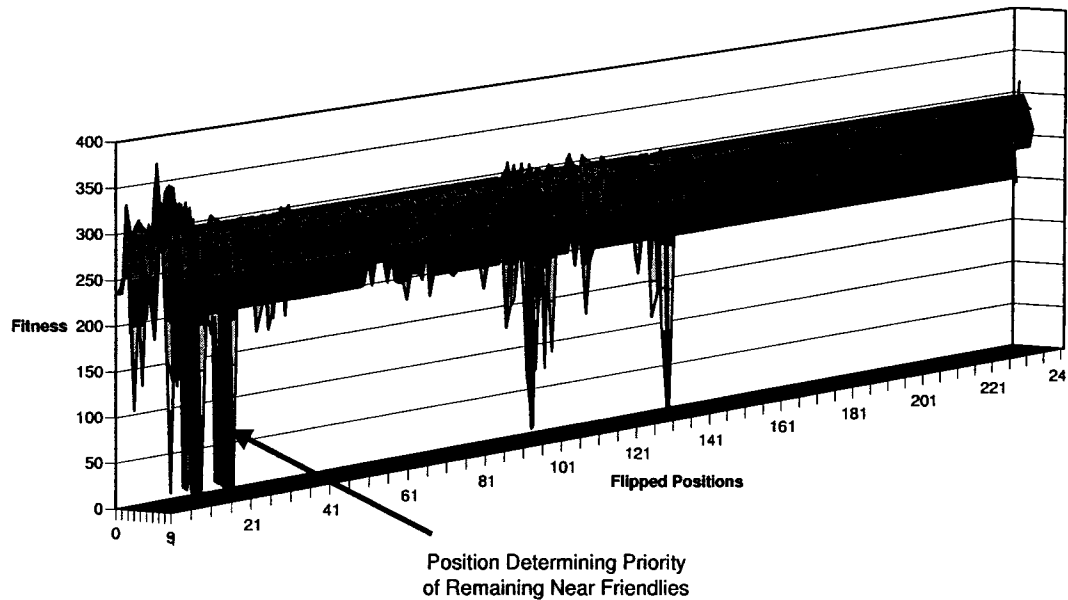
The lack of emphasis on maintaining friendly formation was surprising since it is so counterintuitive to anyone that has conducted ground combat operations. Major emphasis is placed, in training exercises and in combat, on maintaining formation in order to present the most dangerous threat towards the enemy. This counterintuitive result warranted a look at other instances when the emphasis on Rule 1 was higher. The following figure shows the analysis presented in Chapter 4 where areas of insensitivity were punctuated by increasing and decreasing fitness. Looking at the areas of poor fitness, we find that flipping position 17, the position that most increases emphasis on remaining near friendly forces, yields a poor solution across the board. It appears, then, that emphasis on remaining near friendlies may have merits, but overemphasis on that rule is detrimental. In other words, remaining near friendlies cannot win the war, but disproportionate focus devotes excessive effort to “dressing the lines” and not enough effort towards defeating an enemy.

This surprising, yet explainable, result is one more indication that the simulation yields valid results. This increases our confidence in the other results derived.

5.4 Conclusions

The co-evolutionary genetic algorithm confirmed two theoretical expectations: better solutions and wider search. The dynamic modeling technique of testing system capabilities and

Figure 5.1: Effect of Overemphasis on Remaining Near Friendlies.



tactics in an agent-based model demonstrated the ability to develop useful solutions.

Co-evolution resulted in better solutions given an equal number of generations in most of the runs. Although evolution resulted in a similar result since it was searching the same fitness space, evolution took longer to reach that solution.

Co-evolution also maintained a broader set of solutions longer than evolution, confirming the theoretical expectation that co-evolution would allow a wider group of single-step search moves. This is one of the key tenets of the explanation that co-evolution will generally result in better solutions, faster than evolution.

Modeling combat as a complex adaptive system resulted in explainable and appropriate solutions without the intervention of human players. 10's of thousands of generations, with

30 solutions each,⁵ were run, which would have been far beyond the capabilities of human players. The success of this technique recommends it for extension to more complex analysis involving multiple systems and environments.

⁵Although each was not a unique solution.

Chapter 6

The Value of Information

As a general rule the most successful man in life is the man who has the best information.

–Benjamin Disraeli

A pressing question in defense planning is the value of increased information on the battlefield. Nations are spending, literally, billions of dollars to develop and field advanced information technologies designed to speed friendly and enemy information across the force. The underlying assumption is that a force that knows the location and status of friendly and enemy forces will have a distinct advantage over an enemy. This has been described as providing a “step-function” increase in combat capability although little empirical research supports this assertion.

Thus far, measuring the impact of advanced information technology has been problematic. Qualitative assessments of the impact of information technology lead military professionals to believe that the increase is quite dramatic. In fact, many have proposed that combat capabilities might be traded off to pay for these information technologies. The

boldest information technology proponents have proposed, theoretically, that information could replace armor for protection.¹ However, there are no quantitative measurements of that improvement. There are many reasons for this. First, combat simulations that would test impact across a large force aggregates individual systems to the degree that command and control is not measured explicitly. They are singularly unsuited to measuring the impact of a command and control system. Second, most simulation has been done substituting a single type of direct or indirect fire system into known situations to assess improvement. Systems that change the dynamic of how those forces work together are not generally modeled. Current simulations are not suited to this type of modeling for all the reasons discussed in this dissertation; lack of dynamism and inability to search the resultant solution space. Third, the proper tactics required to maximize the impact of those systems, as discussed earlier, is not obvious. Live simulations, like the Advanced Warfighting Experiment in 1998 conducted with an Army Brigade at the National Training Center, showed no such step-function increase in capability, but rather showed in many small ways what the power could be with the proper tactics and training and with the ubiquity of information technology equipment.

Based on the success of coupling an agent-based model with a co-evolutionary genetic algorithm, it was decided to use this approach to explore the value of information. This section describes the approach and its results.

¹This has been a powerful metaphor used by the information technology proponents but thus far there has been no proof that such a trade-off can actually be made.

6.1 Approach

Former Army leaders have described the desired situation as "...knowing where I am, knowing where my friends are, and knowing where the enemy is" [71]. All information known to one system would be shared with all other systems. To simulate this capability, the agent-based model was modified so that any enemy or friendly system detected by one agent appeared to all others. Rather than each agent making decisions based on its local knowledge, each agent made decisions based on global knowledge. An agent's threat state was set by what the entire force could detect, not what the agent could physically detect. All systems had a common threat state on which to make decisions.

Once this modification was made and tested, three runs using random starting solutions were conducted. Each was allowed to run until the set of solutions converged, and then the resulting solutions compared to the original runs that used the co-evolutionary genetic algorithm. The comparisons were made to determine the applicability of this approach and, if successful, to determine the value of information. If a difference could be detected and the results were explainable, it would show that this technique could be used to explore the impact of information technologies and provide some insight into their value, understanding that this simulation is highly simplified. If the approach does show merit, more work with a full complement of different combat and support systems would be warranted.

6.2 Results

Information technology did not raise the overall fitness of the best solutions found in the three runs. The speed at which forces accomplished the mission and avoidance of friendly losses (the basis for the fitness scores) were very similar to the runs outlined in the last chapter, as shown in the following table. The difference came in the increased fitness of previously unfit solutions and the capabilities selected in the most fit solutions.

Table 6.1: Previous Runs v. Information Enhanced Excursion

Runs	High	Average	Low
Previous	391.449	388.587	382.934
Excursion	390.0	388.326	384.987

6.2.1 Increased Fitness

There was a marked increase in the number of solutions that reached high levels of fitness when compared to the non-information enhanced runs. Considering the number of solutions found that reached a fitness of 380.0 (within 5% of the theoretical maximum of 400.0) the results show that information enhancement improved previously less fit solutions and allowed them to compete favorably. Although the excursion runs ran fewer generations, the number of fit solutions found (those with scores over 380.0) was much greater, as shown in Table 6.3.

Table 6.2: Fit Solutions Comparison

Run	# Fit Solutions	# Generations
1	9	9719
2	99	85,546
3	115	16,410
4	30	24,666
5	56	33,645
6	54	30,405
7	53	14,726
8	71	23,793
Exc 1	144	5783
Exc 2	25	5885
Exc 3	132	1783

Running an information-enhanced solution in the agent-based simulation without the increased level of situational awareness resulted in a lowered fitness score, indicating that the solutions were less fit without the ability to share information. This difference provides a measure of the improvement caused by information. The results of using the excursion-selected solutions in the basic, non-information sharing simulation are shown below.

Table 6.3: Comparison of Information-Enhanced Solutions to Standard Information Level

Run	Information Enhanced	Standard Information Level
Excursion 1	390.0	307.1
Excursion 2	385.0	283.9
Excursion 2	390.0	302.4

As can be seen above, information is worth roughly 100 points of fitness to a solution that would not be considered fit in the basic runs. This increase in capability indicates that information increases the utility of otherwise inappropriate solutions.

6.2.2 Information as a Substitute for Capabilities

Careful analysis of the individual solutions selected in the excursion runs shows three differences from the non-information enhanced solutions. Information sharing allows inclusion of less capable sights with which to detect an enemy, inclusion of artillery systems to defeat an enemy at longer range and a reduction in the armor required to protect the tanks. In every excursion run, the solution selected a basic, optical sight with a range of just 2000m and the lowest accuracy of all the potential sight options. This is in stark contrast to the non-information enhanced runs, which all selected a very accurate thermal sight with a range of 5000m. The increased ability of the force to share information on enemy disposition allowed the inclusion of the cheaper sighting system and compensated for the decreased capability.

Two of the three excursion runs included artillery systems in the solutions discovered, also in contradiction to the non-information enhanced runs. The original runs, as reported in the last chapter, selected only solutions that consisted entirely of tanks in order to gain high scores for speed. The two excursion runs divided the available money between tanks and artillery 57/43 and 71/29, respectively. The solutions that waited for indirect fire to be called and take effect were, in essence, penalized in the non-information enhanced runs for their lack of aggressiveness by the speed-emphasizing scoring system. In these excursion runs, however, all of the artillery began firing as soon as the first tank detected an enemy, based on ubiquitous, immediate information. Rapid information sharing allowed the artillery into the fight earlier, negating much of the time penalty exhibited in the base case runs.

The last difference found is that increased information in fact allowed a trade-off of

armor. Two most fit solutions contained tanks with armor of less than 0.2m, and the third allowed tanks with just 0.5m of frontal armor for an average of 0.3m of armor. This is a considerable reduction from the non-information enhanced runs where the armor averaged 0.71m.

The two excursion runs that allowed the lighter armor were the same two that included artillery systems. This indicates that earlier dissemination of the enemy information, resulting in earlier indirect fire, reduced the risk of direct fire engagements and losses to the tank fleet, allowing a decreased level of individual protection. The metaphor of trading information for armor has shown to be more than a simply a figure of speech in this limited simulation.

6.3 Assessment of the Value of Information

The introduction of information sharing in the force does not result in a quantifiable increase in the fitness of already very fit solutions. What it does, however, is increase the fitness of less fit solutions, thereby increasing the variety of fit solutions. This reduces the risk to the force of selecting an inappropriate solution by increasing their applicability and allows solutions to have broader utility.

The value of information can be measured in a specific instance by running a solution through both information enhanced and non-information enhanced simulations. The delta between the fitness scores indicates the value of information in that particular solution.

6.4 Assessment of the Validity of the Approach

Coupling an agent-based model with a co-evolutionary genetic algorithm enabled comparison of solutions with and without information sharing abilities through its matching of tactics to the physical capabilities of the systems. The ability of agent-based models to dynamically adjust to the changing situation makes the approach particularly suited to this type of research and demonstrates the applicability of the approach. Further research with more types of systems is warranted using this approach.

Chapter 7

Further Work

Where is the wisdom we have lost in knowledge?

Where is the knowledge we have lost in information?

—T. S. Eliot

The success of this approach using just two combat systems in a limited scenario indicates the utility of the approach and warrants further exploration with a greater number of systems. This exploration was constrained to two systems as an initial effort, but real decisions cannot be made on that basis. Combat forces are comprised of large numbers of disparate systems such as infantry, engineer and air defense, which should be included in further work.

Increasing the number of candidate systems raises the issue of the appropriate number of crossover points. Co-evolution using two-point crossover was appropriate with two systems, but increased numbers of systems may benefit from multiple-point crossover schemes that allow all systems to simultaneously co-evolve.

This research used a single scenario to determine efficacy of the approach, but a method to test solutions in multiple scenarios, with multiple types of terrain, enemy systems and missions, should be explored. This would allow full testing of solutions and prevent selection of “brittle” solutions-only appropriate in a single prescribed instance.

Agent-based modeling showed value with a small-scale, tactical-level force. Determination of the value of this modeling technique when evaluating higher-level organizations appears warranted. Combat interactions increase in complexity and our ability to replicate them with current linear models decreases as forces become larger, indicating that an agent-based approach would be more useful at higher levels than current aggregated models.

Increased information sharing showed great value and the approach appeared to capture those benefits. This opens an entire research area waiting to be explored now that an appropriate tool is available.

Chapter 8

Conclusion

I know that most men, including those at ease with problems of the greatest complexity, can seldom accept even the simplest and most obvious truth if it be such as would oblige them to admit the falsity of conclusions which they have delighted in explaining to colleagues, which they have proudly taught to others, and which they have woven, thread by thread, into the fabric of their lives.

–Leo Tolstoy

8.1 Intent of the Dissertation

This dissertation intended to provide a solution for one of the most vexing problems in the government, force development of combat systems. By the nature of the decisions involved, they commit large sums of money, encompass a wide universe of types and capabilities of equipment, and have long-term consequences. Although the decisions, at their most basic level, are to develop the most capable force for an affordable cost, it can never be forgotten that the decisions are critical to the long-term well-being of the nation as well as the very

lives of the men and women that use the fielded systems. Given this significance and the complexity of the choices available, the solution of this problem is a significant step forward for the Department of Defense as well as other government agencies that can adapt this approach for their uses.

The force development problem suffers from two distinct but intertwined problems. First, combat is highly non-linear and dynamic. Small inputs can have no result whatsoever until reaching a critical mass, then the marginal impact of increased input can be significant. Once a saturation level is reached, marginal impact flattens or even turns negative. Combat is very situational; decisions must be appropriate in place, force capabilities and time. Change in the timing of force movement or the number of systems that reach a decision point, requires a change in either the substance or timing of the decision. Current combat models are unsuited to replicate this non-linearity except when humans are intimately involved throughout every stage of the simulation as players.

Second, the large number of types and capabilities of equipment drive the number of potential solutions far beyond what can be explored with man-in-the-loop processes. Solution sets of 10^{60} unique solutions are not unusual given the number of individual choices for each combat system available. Further, even if a search could be made using humans to react to changing situations, the most appropriate tactics are not always obvious when dealing with new technologies or applying them in different ways.

This dissertation determined to explore the ability of an agent-based model to dynamically model potential solutions for fitness in a combat environment. In order to search the solution space, a co-evolutionary genetic algorithm was evaluated as a potential improve-

ment over a standard, evolutionary genetic algorithm.

8.2 Restatement of the Problem and Approach

This problem can be abstracted to a stochastic, mixed-integer, non-linear optimization problem with a very large solution set. No closed form representation of an optimization equation is available, requiring a derivative-free solution method and a search method that can accommodate a hyperdimensional solution set.

To solve the force development problem, two approaches needed to be developed and explored; a dynamic modeling technique and an appropriate search technique. To model the non-linearities of combat, the underlying assumption on which existing models were built were reexamined. Current models aggregate the interactions of lower units, losing the essential dynamic of combat, the human factor. Instead of armies being simulated as a collection of independent agents working together based on an awareness of the mission and capabilities, whose higher-level performance emerges from the myriad interactions, they become monolithic entities that perform in predetermined ways. This dissertation developed an agent-based model that incorporated both physical capabilities and tactical rules that determine the agents' actions. In this way, not only the appropriateness of the equipment was measured in the simulation, but based that on the most appropriate tactics. The combination of capability and tactics resulted in an overall fitness of each tested solution.

To search this solution space, this dissertation developed a classification method for landscapes described by binary-coded problem representations. Landscape shapes have

been described in various terms, but without a great deal of rigor or comparability. Landscape shape is a function of the problem to be solved, as well as its representation for solution. Representation as a binary-coded string is common. This dissertation developed a standardized way to measure and categorize the resulting landscapes.

To explore the advantages of co-evolution, a rigorous theoretical underpinning for a co-evolutionary approach was developed. This dissertation proved that there are advantages to co-evolution, namely increased probability that good schemata would survive the crossover process and that the increased available step size at each generation would allow faster search across the landscape. These advantages were alluded to in previous works and even explained in a naturalistic manner, but the improvement offered had not been previously proven.

When the theory recommended pursuing this approach, two combat systems were allowed to evolve simultaneously between generations in a dynamic, agent-based model. In each run, force capability was maximized given a fixed amount of money available to the force. Solutions selected from a menu of capabilities ranging from engines, armor protection, sights and weapons. Tactics were governed by changing the priority of three rules, remaining in formation with other friendly systems, moving to attack a perceived enemy and moving to the objective. Artillery systems had an additional rule that governed where they moved in relation to the tanks.

8.3 Results

Coupling these powerful techniques resulted in a useful method to find very-fit solutions. The nature of the landscape was such that many very fit solutions were available. By making multiple runs and comparing the solutions found, it became very clear what the high-value capabilities were which would allow decision makers a guide to making capability trade-offs.

A recap of the findings:

- In six of eight record runs, co-evolution found solutions with higher fitness than an evolutionary approach.
- In seven of eight runs, co-evolution continued to search a wider set of solutions well after evolution had converged on a most-fit solution. This increase is attributable to the larger step available in each generation and increases the opportunity to find more fit solutions.
- A good, relatively inexpensive sight was adequate. In fact, all solutions selected the same sight, attesting its high benefit:cost ratio.
- Armor protection was useful, but only up to a threshold. Beyond that it became a detriment. Armor protecting the side and top of vehicles was important enough to pay a weight and expense penalty.
- Low cost conventional guns were selected over more expensive, but more accurate, missiles and electromagnetic guns.
- Lighter armor allowed selection of a cheap engine with limited power.

- Tactics that favored aggressiveness to get to the objective and proper alignment of forces in relationship to the enemy were very important. Alignment of friendly forces seemed to have little impact on the fitness of a solution.

8.4 Measurement of the Value of Information

As an excursion, the force was adapted to share information instantaneously to determine if the value of information could be measured. The availability of information showed little improvement in already highly fit solutions. However, it increased the fitness of previously less fit solutions making a wider universe of solutions acceptable. This equates to reducing the risk of a force decision, since more solutions are relatively equal in fitness values. The danger of making an inappropriate decision is lessened by the leveling effect of shared information.

8.5 Implications for Future Work

This dissertation fulfilled its intended goals by developing a dynamic model to simulate combat, and developing the theoretical framework for and showing the utility of co-evolution to search the solution space. The approach developed to solve the force development problem is a significant step forward from current methods. This approach, which used only two combat systems in a single combat scenario for research purposes, now needs to be expanded to include the multitude of equipment that could be found in combat units and

more, representative scenarios need to be developed in order to test solutions across the spectrum of combat to prevent selection of brittle solutions of limited utility.

Appendix A

Code for the Agent-based Model

The following code was used to build the Agent-based model developed for this dissertation.

The model is called the “MULE” in no small part because of the Army mascot. The files are listed below with the header files. A sample input file is provided to enable someone to compile the code and run a test using redirection.

```

////////////////////////////////////
//                                                                    //
//  Main.cpp: Provides the control for the Mule simulation  //
//                                                                    //
////////////////////////////////////

#include "vehmgr.h"
#include "land.h"
#include "Param.h"
#include "RParam.h"
#include "rngs.h"
#include "rvgs.h"
#include "rvms.h"
#include <stdio.h>
#include <iostream>
#include <ctime>
#include <string>
#include <cstdlib>
#include <cmath>

#define LOC 0.95

using namespace std;

double Score(long B_Start);           //declarations
long Cleanup(void);

extern World Cell[Landscape_Size][Landscape_Size];

double Cur_Time = 0.0;

Veh* BT = NULL;           //Pointer to Blue Tanks
Veh* BA = NULL;           //Pointer to Blue Arty

Veh* RT = NULL;           //Pointer to Red Tanks
Veh* RA = NULL;           //Pointer to Red Arty

ArTGT* Imp = NULL;       //Pointer to arty impacts

long BNum = 0;           //Number of Blue Vehicles
long RNum = 0;           //Number of Red Vehicles

long BTNum = 0;          //Number of Blue Tanks
long BANum = 0;          //Number of Blue Arty

```

```

long   RTNum = 0;           //Number of Red Tanks
long   RANum = 0;         //Number of Red Arty

char genome[243];         //Accepts the soln string

double N[5];             //contains next event times

int main()
{
    long seed;

    PutSeed(54778);
    GetSeed(&seed);

    long Next_Event;      //next-event list
    // long count = 0;
    cin.getline(genome, 243);

    double w = 100.0;
    long n = 0;
    double sum = 0.0;
    double mean = 0.0;
    double data;
    double stdev;
    double u, t;
    double diff;
    clock_t wait = 5 * CLOCKS_PER_SEC + clock();

    while(w > 5.0 && n < 100 && clock() < wait) {

        long B_Start = 0;    //counts number of Blue veh's at start

        Init_World();       //initializes the landscape

        BT = Init_BT(genome, 243);    //initializes blue tanks
        BA = Init_BA(genome, 243);    //initializes blue arty
        RT = Init_RT(genome, 243);    //initializes red tanks
        RA = Init_RA(genome, 243);    //initializes red arty

        B_Start = BNum;

        if(BT != NULL) Put_Vehs(BT); //places Blue Tanks on the landscape
        if(BA != NULL) Put_Vehs(BA); //places Blue Arty on the landscape
        Put_Vehs(RT);                //places Red Tanks on the landscape
        Put_Vehs(RA);                //places Red Arty on the landscape
    }
}

```

```

enum NE {BTank, BArt, RTank, RArt, Impact};

if(BT != NULL) N[BTank] = BT->NextTime.N;
else N[BTank] = Inf; //init event schedule
if(BA != NULL) N[BArt] = BA->NextTime.N;
else N[BArt] = Inf;
N[RTank] = RT->NextTime.N;
N[RArt] = RA->NextTime.N;
N[Impact] = Inf;

while ( (Cur_Time < Time_Limit)&&(BTNum+BANum>0)&&(RTNum>0) )
{
Next_Event = BTank;
Cur_Time = N[BTank];

if(Cur_Time > N[BArt]) {Next_Event = BArt; Cur_Time = N[BArt];}
if(Cur_Time > N[RTank]) {Next_Event = RTank; Cur_Time = N[RTank];}
if(Cur_Time > N[RArt]) {Next_Event = RArt; Cur_Time = N[RArt];}
if(Cur_Time > N[Impact]) {Next_Event = Impact; Cur_Time = N[Impact];}

switch (Next_Event)
{
case BTank : BT = ProcTk(BT);
N[BTank] = BT->NextTime.N;
break;

case BArt : BA = ProcArt(BA);
N[BArt] = BA->NextTime.N;
break;

case RTank : RT = ProcTk(RT);
N[RTank] = RT->NextTime.N;
break;

case RArt : RA = ProcArt(RA);
N[RArt] = RA->NextTime.N;
break;

case Impact : Imp = ProcImp(Imp);
if(Imp != NULL) N[Impact] = Imp->time;
else N[Impact] = Inf;
break;

default : cerr << "prob in main" << endl;

```



```

}

if(BT == NULL || BA == NULL || RT == NULL || RA == NULL)
    //if any are null, set next
{
    //time to inf
    if(BT == NULL) N[BTank] = Inf;
    if(BA == NULL) N[BArt] = Inf;
    if(RT == NULL) N[RTank] = Inf;
    if(RA == NULL) N[RArt] = Inf;
}
}

data = Score(B_Start);
//cout << data << endl;

n++;
diff = data - mean;
sum += diff * diff * (n - 1.0) / n;
mean += diff / n;
stdev = sqrt(sum / n);
if(n>1) {
    u = 1.0 - 0.5 * (1.0 - LOC);
    t = idfStudent(n-1, u);
    w = t * stdev / sqrt(double(n-1));
}
Cleanup();
}
cout << mean << endl;
return 0;
}

```

```

double Score(long B_Start)
{
    double Time;
    double Score;
    long B_Surv;
    double PctSurv;

    if(Cur_Time >= Time_Limit) return 0.0;
    else {
        Time = Time_Limit - Cur_Time;
        B_Surv = B_Start - BNum;
        PctSurv = double(B_Surv)/double(B_Start);
        Score = Time + PctSurv*200.0;
        //cout << Score << endl;
    }
}

```

```

    }
    return Score;
}

long Cleanup(void)
{
    Cur_Time = 0.0;           //resets time
    Veh* Temp = NULL;        //Temp pointers
    ArTGT* TempImp = NULL;

    while(BT != NULL) {     //Cleans Pointer to Blue Tanks
        Temp = BT;
        BT = BT->Next;
        delete Temp;
        Temp = NULL;
    }

    while(BA != NULL) {     //Cleans Pointer to Blue Arty
        Temp = BA;
        BA = BA->Next;
        delete Temp;
        Temp = NULL;
    }

    while(RT != NULL) {     //Cleans Pointer to Red Tanks
        Temp = RT;
        RT = RT->Next;
        delete Temp;
        Temp = NULL;
    }

    while(RA != NULL) {     //Pointer to Red Arty
        Temp = RA;
        RA = RA->Next;
        delete Temp;
        Temp = NULL;
    }

    while(Imp != NULL) {    //Cleans pointer to arty impacts
        TempImp = Imp;
        Imp = Imp->Next;
        delete TempImp;
        TempImp = NULL;
    }
}

```

```
BNum = 0;           //Number of Blue Vehicles
RNum = 0;           //Number of Red Vehicles

BTNum = 0;         //Number of Blue Tanks
BANum = 0;         //Number of Blue Arty

RTNum = 0;         //Number of Red Tanks
RANum = 0;         //Number of Red Arty
return 0;
}
```

```

////////////////////////////////////
//                                                                    //
// Veh.CPP: Member functions of the Vehicle class.                    //
//                                                                    //
////////////////////////////////////

#define STRICT
#define sqr(x)    ((x) * (x))
#include "veh.h"
#include "vehmgr.h"
#include "rvgs.h"
#include "rngs.h"
#include "land.h"
#include "Param.h"
#include "RParam.h"
#include <math.h>
#include <iostream>
using namespace std;

extern World Cell[Landscape_Size][Landscape_Size];

extern long BNum;
extern long BNum;

extern long RNum;
extern long RNum;

extern long BEngine;
extern long BAmmo_Type;
extern long BAmmo_Qty;
extern long BSight;
extern long BAuto;
extern long BArmor;

////////////////////////////////////
// Constructors, destructors, and overloaded operators:              //
////////////////////////////////////

// default constructor:
Veh::Veh(long x, long y, char clr, char arr[], int n)
{
    extern double Engine[4][4];
    extern double Ammo[14][10];
    extern double Sight[4][6];

```

```

extern double Autoloader[3];

double Fix_Force = Conv(arr, n, 98, 102) * 0.06666;
//pct of force fixing enemy
double p; //used to draw probs
double wt, amtarmor, cubes, spd;

X = x;
Y = y;
Dest_X = x; //first move is in place
Dest_Y = y;
Last_X = x; //last cell visited to
//damp oscillation
Last_Y = y;

if(clr == 'b') //veh is blue
{
  ObjX = XObj; //carry obj coord
  ObjY = YObj;

  Dir = 0; //blue face N, red face S

  //select vision
  SelectStream(VISION_STREAM);
  Vision = long(( Sight[BSight][5] +
    Equilikely(-long(Sight[BSight][5]/10),
    long(Sight[BSight][5]/10) ))/25);

  pd = Sight[BSight][1];

  //compute cubes then length
  cubes = Engine[BEngine][2] + 5*BAMmo_Qty*Ammo[BAMmo_Type][2] +
    Ammo[BAMmo_Type][7] + Sight[BSight][2] + BAuto*Autoloader[1] + 20.0;

  l = cubes/(2.15*(2.8-(BAuto*0.6)));
  h = 2.8-(BAuto*0.6);
  w = 3.5;

  //compute weight -> speed
  amtarmor = 2*(2.8-(BAuto*0.6))*2.15*(BArmor*0.1);
//m^3 of armor on tank
//2* front slab*armor thickness

  wt = amtarmor*Arm_Wt + Engine[BEngine][0] + (5 * BAMmo_Qty *
    Ammo[BAMmo_Type][0]) + Ammo[BAMmo_Type][6] + Sight[BSight][0] +

```

```

    (BAuto*Autoloader[0]);

    spd = 6.49 + 1.49*Engine[BEngine][1]/wt; //converts hp/t to km/hr

    SelectStream(M_RATE_STREAM);
    Move_Rate = Equilikely(long(0.9*spd),long(1.1*spd))*40.0/60.0;

    Type = 0; //blue tanks are type 0, red are type 2
    Power = 1.0; //cbt effectiveness at start (100%)
    Color = clr; //set color
    G_Rg = long(Ammo[B Ammo_Type][4]/25);
    //G_Rg is in grids, not meters
    acc = Ammo[B Ammo_Type][5];
    //acc of gun/ammo comb
    stacc = Sight[BSight][4]; //acc of sight
    Armor = BArmor*0.1; //thickness of frontal armor
    Pen = Ammo[B Ammo_Type][1];
    //penetration cap of bullets
    Moving = 1; //starts not moving
    State = 0; //blue starts on offense
    Rds = B Ammo_Qty * 5; //number of rounds on board
    enctr = 0; //initially can't see any en
    frctr = 1; //can always see self
    arctr = 0; //
    EnDir = 8; //cannot see enemy, so doesn't have a
    // perceived direction
    CFF = 1.0/(Move_Rate * CFF_Min );
    //sets prob of a call for fire

    Reload = Ammo[0][9];

    SelectStream(FIX_FORCE_STREAM);
    p = Uniform(0,1);
    if(p < Fix_Force) Fix = 0; //part of fixing force
    else Fix = 1; //part of maneuver force

    NextTime.N = 0.0; //init time of next events
    NextTime.Sh = Inf;
    NextTime.Mv = 0.0;

    Next = NULL;
    Fr = NULL;
    Arty = NULL;
    En = NULL;

```

```
Target = NULL;
tgtctr = 0;
Shots_Msn = 0;
Width    = 0;

Nof = Conv(arr, n, 17, 20);
Noa = 0;
Noe = Conv(arr, n, 20, 23);
Noo = Conv(arr, n, 23, 26);

ALof = Conv(arr, n, 26, 29);
ALoa = 0;
ALoe = Conv(arr, n, 29, 32);
ALoo = Conv(arr, n, 32, 35);

AMedf = Conv(arr, n, 35, 38);
AMeda = 0;
AMede = Conv(arr, n, 38, 41);
AMedo = Conv(arr, n, 41, 44);

AHif = Conv(arr, n, 44, 47);
AHia = 0;
AHie = Conv(arr, n, 47, 50);
AHio = Conv(arr, n, 50, 53);

APanf = Conv(arr, n, 53, 56);
APana = 0;
APane = Conv(arr, n, 56, 59);
APano = Conv(arr, n, 59, 62);

DLOf = Conv(arr, n, 62, 65);
DLOa = 0;
DLOe = Conv(arr, n, 65, 68);
DLOo = Conv(arr, n, 68, 71);

DMedf = Conv(arr, n, 71, 74);
DMeda = 0;
DMede = Conv(arr, n, 74, 77);
DMedo = Conv(arr, n, 77, 80);

DHif = Conv(arr, n, 80, 83);
DHia = 0;
DHie = Conv(arr, n, 83, 86);
DHio = Conv(arr, n, 86, 89);
```

```

DPanf = Conv(arr, n, 89, 92);
DPana = 0;
DPane = Conv(arr, n, 92, 95);
DPano = Conv(arr, n, 95, 98);

O_Dist = Conv(arr, n, 102, 105);

}

else //tank is red, inputs hard-coded
{
  ObjX = XObj; //carry obj coord
  ObjY = YObj;

  Dir = 4; //blue face N, red face S

  SelectStream(VISION_STREAM);
  Vision = Equilikely(2750/25, 2250/25);
  pd = 0.6;

  h = 2.3;
  l = 6.0;
  w = 3.5;

  SelectStream(M_RATE_STREAM);
  Move_Rate = Equilikely(22, 18);
  //30 kph +/- 10%

  Type = 2; //red are type 2
  Power = 1.0; //cvt effectiveness at start (100%)
  Color = clr; //set color
  G_Rg = Equilikely(90, 110);
  //G_Rg is 2500m in grids

  acc = 1.0; //acc of gun/amo comb
  stacc = 1.0; //acc of sight
  Armor = 0.520; //thickness of frontal armor
  Pen = 0.550; //penetration cap of bullets
  Moving = 1; //starts not moving
  State = 5; //red starts on def
  Rds = 40; //number of rounds on board
  enctr = 0; //initially can't see any en
  frctr = 1; //can always see self
  arctr = 0; //
  EnDir = 8; //cannot see enemy, so doesn't have a
  // perceived direction

```



```
CFF = 1.0/( double(Move_Rate * CFF_Min) );
//sets probab of a call for fire

Reload = 5;

Fix = 0; //part of fixing force

NextTime.N = 0.0; //init time of next events
NextTime.Sh = Inf;
NextTime.Mv = 0.0;

Next = NULL;
Fr = NULL;
Arty = NULL;
En = NULL;

Target = NULL;
tgtctr = 0;
Shots_Msn = 0; //used in art.cpp
Width = 0; //used in art.cpp

Nof = 1;
Noa = 0;
Noe = 0;
Noo = 1;

ALof = 1;
ALoa = 0;
ALoe = 5;
ALoo = 1;

AMedf = 3;
AMeda = 0;
AMede = 10;
AMedo = 1;

AHif = 1;
AHia = 0;
AHie = 10;
AHio = 1;

APanf = 0;
```

```

    APana = 0;
    APane = 10;
    APano = 0;

    DLoF = 1;
    DLoa = 0;
    DLoe = 0;
    DLoo = 2;

    DMedf = 3;
    DMeda = 0;
    DMede = 0;
    DMedo = 2;

    DHif = 1;
    DHia = 0;
    DHie = 1;
    DHio = 1;

    DPanf = 0;
    DPana = 0;
    DPane = 1;
    DPano = 0;

    O_Dist = 8;           //opt dist = 200m
}

// destructor:
Veh::~Veh ()
{
    ClearPtrs();
    if(Color== 'r') RNum--;
    else BNum--;
}

////////////////////////////////////
// Other member functions:           //
////////////////////////////////////

bool Veh::Choose_Next_Move(long f, long e, long o)
{
    double dist, distf, diste, disto, best;
                               //dist to dest, fr, en, obj and best pri

```

```

long fx, fy, fx1, fy1, ex, ey, ox, oy, a, i;
double pri[3];          //holds priority calc

if(NextTime.Sh == Inf && State > 0) NextTime.Sh = NextTime.N + Reload;
                        //schedule a shot

TGT* T1 = NULL;
TGT* T2 = NULL;

//find best location based on friendlies
switch (frctr)
{
case 1 : fx = X;          //no other fr are in sight
        fy = Y;
        break;

case 2 : Fr_Locn(Fr->X, Fr->Y, fx, fy); //just one other fr in sight
        break;

default: T1 = Fr;        //mult fr's in sight
        T2 = Fr->Next;
        Fr_Locn(T1->X, T1->Y, fx, fy);
        Fr_Locn(T2->X, T2->Y, fx1, fy1);
        fx = Rnd((fx+fx1)/2.0);
        fy = Rnd((fy+fy1)/2.0);
        T1 = NULL;
        T2 = NULL;
}

distf = Dist(X, Y, fx, fy); //find dist and priority of movement
pri[0] = f * distf;        //based on friendlies

//find best locn based on closest enemy

switch(enctr)
{
case 0 : ex = X;          //no en in sight
        ey = Y;
        break;

default : En_Locn(En->X, En->Y, ex, ey);
}
diste = Dist(X, Y, ex, ey);
pri[1] = e * diste;

```

```

//find best locn based on obj

Obj_Locn(ObjX, ObjY, ox, oy);
disto = sqrt(Dist(X, Y, ObjX, ObjY));
pri[2] = o * disto;
disto = sqr(disto);
//best location based on highest pri of the three:

a = 0;
best = pri[0];
for(i=1; i<3; i++)           //find highest priority move
{
    if(pri[i] > best)
{
    best = pri[i];
    a = i;
}
}

switch(a)
{
    case 0: if(distf != 0.0)
        {
Dest_X = Rnd(X + (fx - X)/distf);
Dest_Y = Rnd(Y + (fy - Y)/distf);
        }
    else
        {
Dest_X = X;
Dest_Y = Y;
        }
    break;

    case 1:
        Dest_X = Rnd(X + (ex - X)/diste);
        Dest_Y = Rnd(Y + (ey - Y)/diste);
        break;

    case 2:
        Dest_X = Rnd( X + (ox - X)/disto );
        Dest_Y = Rnd( Y + (oy - Y)/disto );
        break;

    default: cerr << "problem in ch_best_move" << endl;
}

```

```

if(Dest_X == Last_X && Dest_Y == Last_Y)
    //if moving back to previous spot, don't
    {
        Dest_X = X;
        Dest_Y = Y;
    }

//set moving flag
if(Dest_X != X || Dest_Y != Y) Moving = true;
else Moving = false;

//set direction
//if moving point direction of travel

if(Moving == true) Set_Dir(X - Dest_X + Dest_Y - Y);

//if stationary and enemy in sight, point at closest enemy
else if(En != NULL)
    {
        diste = Dist(X, Y, En->X, En->Y);
        Set_Dir( Rnd((X - En->X)/diste) + Rnd((Y - En->Y)/diste) );
    }

//if no enemy, assume default direction
else
    {
        if(Color == 'r') Dir = 4;
        else Dir = 0;
    }

//determine next update time
dist = Dist(X, Y, Dest_X, Dest_Y);

//if sitting on best spot, stay 1/shots per min
if(dist == 0.0) NextTime.Mv = NextTime.Mv + Reload;

//else compute next event time
else NextTime.Mv = NextTime.Mv + (dist/Move_Rate);
return true;
}

double Veh::Fr_Locn(long frX, long frY, long &fx, long &fy)
{
    double p;

```

```

p = Uniform(0,1);

//best locn is offset from the friendly by the opt dist
if( (frX > X) || ((frX == X) && (p < 0.5)))
    fx = Rnd( double(frX) - double(O_Dist) ); //fr is below
else fx = Rnd( double(frX) + double(O_Dist)); //else above

p = Uniform(0,1);
if( (frY > Y) || ((frY == Y) && (p < 0.5)))
    fy = Rnd( double(frY) - double(O_Dist) ); //fr is right
else if(frY < Y) fy = Rnd( double(frY) + double(O_Dist) );//or left
else fy = frY; //else on-line

fx = max(0, fx); //stay on game board
fy = max(0, fy);
fx = min(fx, Landscape_Size-1);
fy = min(fy, Landscape_Size-1);
return 1.0;
}

double Veh::En_Locn(long enX, long enY, long &ex, long &ey)
{
    long x1, x2, y1, y2; //temp locations
    double dist = pow(10, -.07918)*G_Rg; //best dist from en
    double diste = Dist(X, Y, enX, enY); //actual dist from en
    long ls = Landscape_Size-1;
    if(Fix == 1) //part of atk'g maneuver force
    {
        switch(EnDir)
        {
        case 0: if( 0.9*dist < diste && diste < 1.1*dist &&
            ( enX-X <= 0 || enX-X <= mabs(Y-enY)) )
            { //if in proper range, don't move
                ex = X;
                ey = Y;
            }
        else
        {
            ex = enX;
            y1 = enY - Rnd(dist); //en pointed north, move to
            y2 = enY + Rnd(dist); //flank
            if(y1<0) ey = y2; //x component not on game bd
            else if(y2>ls) ey = y1;
            else if(Dist(X, Y, ex, y1)<Dist(X, Y, ex, y2)) ey = y1; //ex,y1 closer
            else ey = y2;
        }
    }
}

```

```

    }
    break;

    //ex,y2 closer

case 1: if( 0.9*dist < diste && diste < 1.1*dist &&
    (enX < X || Y < enY) )
    {
        //if in proper range, don't move
        ex = X;
        ey = Y;
    }
else
    {
        x1 = enX - Rnd(sqrt(sqr(dist)/2));
        y1 = enY - Rnd(sqrt(sqr(dist)/2));
        x2 = enX + Rnd(sqrt(sqr(dist)/2));
        y2 = enY + Rnd(sqrt(sqr(dist)/2));
        if(x1 < 0 || y1 < 0) //x1 or y1 not on game bd
            {
                ex = x2;
                ey = y2;
            }
        //x2 or y2 not on game bd
        else if(x2 > ls || y2 > ls)
            {
                ex = x1;
                ey = y1;
            }
        else if(Dist(X, Y, x1, y1)<Dist(X, Y, x2, y2)) //x1,y1 closer
            {
                ex = x1;
                ey = y1;
            }
        //x2,y2 closer
        else
            {
                ex = x2;
                ey = y2;
            }
    }
break;

case 2: if( 0.9*dist < diste && diste < 1.1*dist &&
    (Y-enY <= 0 || Y-enY <= mabs(X-enX)) )
    {
        //if in proper range, don't move
        ex = X;
        ey = Y;
    }
else

```

```

{
  x1 = enX - Rnd(dist);           //en pointed east, move to
  x2 = enX + Rnd(dist);           //flank
  ey = enY;

  if(x1<0)   ex = x2;              //x component not on game bd
  else if(x2>ls) ex = x1;
  else if(Dist(X, Y, x1, ey)<Dist(X, Y, x2, ey)) ex = x1; //x1,ey closer
  else ex = x2;
}                                  //x2,ey closer
break;

case 3: if( 0.9*dist < diste && diste < 1.1*dist &&
  ( X < enX || Y < enY ) )
  {
    //if in proper range, don't move
    ex = X;
    ey = Y;
  }
else
  {
    x1 = enX + Rnd(sqrt(sqr(dist)/2));
    y1 = enY - Rnd(sqrt(sqr(dist)/2));
    x2 = enX - Rnd(sqrt(sqr(dist)/2));
    y2 = enY + Rnd(sqrt(sqr(dist)/2));
    if(x1 > ls || y1 < 0)           //x1 or y1 not on game bd
      {
ex = x2;
ey = y2;
      }
    else if(x2 < 0 || y2 > ls)     //x2 or y2 not on game bd
      {
ex = x1;
ey = y1;
      }
    else if(Dist(X, Y, x1, y1)<Dist(X, Y, x2, y2)) //x1,y1 closer
      {
ex = x1;
ey = y1;
      }
    else                             //x2,y2 closer
      {
ex = x2;
ey = y2;
      }
  }
}

```



```

break;

case 4: if( 0.9*dist < diste && diste < 1.1*dist &&
  (X-enX <= 0 || X-enX <= mabs(Y-enY)) )
  {
    //if in proper range, don't move
    ex = X;
    ey = Y;
  }
else
  {
    ex = enX;
    y1 = enY - Rnd(dist); //en pointed north, move to
    y2 = enY + Rnd(dist); //flank
    if(y1<0) ey = y2; //x component not on game bd
    else if(y2>ls) ey = y1;
    else if(Dist(X, Y, ex, y1)<Dist(X, Y, ex, y2)) ey = y1; //ex,y1 closer
    else ey = y2;
  } //ex,y2 closer
break;

case 5: if( 0.9*dist < diste && diste < 1.1*dist &&
  ( X < enX || enY < Y ) )
  {
    //if in proper range, don't move
    ex = X;
    ey = Y;
  }
else
  {
    x1 = enX - Rnd(sqrt(sqr(dist)/2)); //en pointed sw
    y1 = enY - Rnd(sqrt(sqr(dist)/2));
    x2 = enX + Rnd(sqrt(sqr(dist)/2));
    y2 = enY + Rnd(sqrt(sqr(dist)/2));
    if(x1 < 0 || y1 < 0) //x1 or y1 not on game bd
      {
ex = x2;
ey = y2;
      } //x2 or y2 not on game bd
    else if(x2 > ls || y2 > ls)
      {
ex = x1;
ey = y1;
      }
    else if(Dist(X, Y, x1, y1)<Dist(X, Y, x2, y2)) //x1,y1 closer
      {
ex = x1;

```

```

ey = y1;
  }
  else //x2,y2 closer
  {
ex = x2;
ey = y2;
  }
}
break;

case 6: if( 0.9*dist < diste && diste < 1.1*dist &&
  (enY-Y <= 0 || enY-Y <= mabs(X-enX)) )
  { //if in proper range, don't move
    ex = X;
    ey = Y;
  }
else
  {
    x1 = enX - Rnd(dist); //en pointed east, move to
    x2 = enX + Rnd(dist); //flank
    ey = enY;
    if(x1<0) ex = x2; //x component not on game bd
    else if(x2>ls) ex = x1;
    else if(Dist(X, Y, x1, ey)<Dist(X, Y, x2, ey)) ex = x1; //x1,ey closer
    else ex = x2; //x2,ey closer
  }
break;

case 7: if( 0.9*dist < diste && diste < 1.1*dist &&
  ( enX < X || enY < Y ) )
  { //if in proper range, don't move
    ex = X;
    ey = Y;
  }
else
  {
    x1 = enX + Rnd(sqrt(sqr(dist)/2));
    y1 = enY - Rnd(sqrt(sqr(dist)/2));
    x2 = enX - Rnd(sqrt(sqr(dist)/2));
    y2 = enY + Rnd(sqrt(sqr(dist)/2));
    if(x1 > ls || y1 < 0) //x1 or y1 not on game bd
    {
ex = x2;
ey = y2;
    }
  }

```

```

        else if(x2 < 0 || y2 > ls)           //x2 or y2 not on game bd
        {
ex = x1;
ey = y1;
        }
        else if(Dist(X, Y, x1, y1)<Dist(X, Y, x2, y2)) //x1,y1 closer
        {
ex = x1;
ey = y1;
        }
        else                                     //x2,y2 closer
        {
ex = x2;
ey = y2;
        }
    }
break;

default: cerr << "en_locn prob" << ' ' << Color << ' ' << X << ' '
        << Y << ' ' << NextTime.Mv << endl;
}
    }//end if(Fix == 1)

    else                                     //atk'g but part of fixing force, or defending
    {
        if(0.9*dist<=diste && 1.1*dist<=diste)//in range band, don't move
    {
ex = X;
ey = Y;
    }
        else                                     //move to proper range
    {
ex = enX + Rnd((dist/diste)*(X-enX));
ey = enY + Rnd((dist/diste)*(Y-enY));
    }
        }

return 1.0;
}

double Veh::Obj_Locn(long obX, long obY, long &ox, long &oy)
{
double dist = Dist( X, Y, obX, obY);
if(dist != 0.0)
    {

```

```

        ox = obX;                //obj attracts tank
        oy = obY;
    }
    else
    {
        ox = obX;
        oy = obY;
    }
    ox = max (0, ox);            //stay on game board
    oy = max (0, oy);
    ox = min (ox, Landscape_Size-1);
    oy = min (oy, Landscape_Size-1);
    return 1.0;
}

bool Veh::MoveTo(long mX, long mY) //places tank in new location
{
    Last_X = X;                //records where tank moved from
    Last_Y = Y;
    X = mX;                    //updates new loaction
    Y = mY;
    return true;
}

bool Veh::PutFr(long fr) //stores number of frdly, arty and en in area
{
    frctr = fr;
    return true;
}

bool Veh::PutAr(long ar)
{
    arctr = ar;
    return true;
}

bool Veh::PutEn(long en)
{
    enctr = en;
    return true;
}

double Veh::Ratio()
{
    return (double(enctr)/frctr);
}

```

```

}

bool Veh::SetState(long x)
{
    State = x;
    return true;
}

bool Veh::ClearPtrs()
{
    TGT* temp;
    temp = Fr;
    while (temp != NULL)                //there were fr's in area
    {
        Fr = Fr->Next;
        delete temp;
        temp = Fr;
    }
    temp = En;
    while (temp != NULL)                //there were en's in area
    {
        En = En->Next;
        delete temp;
        temp = En;
    }
    temp = Arty;
    while(temp != NULL)
    {
        Arty = Arty->Next;
        delete temp;
        temp = Arty;
    }
    enctr = 0;
    frctr = 1;
    return true;
}

bool Veh::Set_Dir(long D) //set the direction flag
{
    D=D+2;
    switch (D)
    {
        case 0: Dir = 5;           //if (Dest-Loc'n)+2 == 0, must be going SW
            break;
    }
}

```

```

    case 1: if(Dest_X == X) Dir = 6;      // dir flags: 7 0 1
    else Dir = 4;                        //           6  2
    break;                               //           5 4 3

    case 2: if(Dest_X < X) Dir = 7;
    else Dir = 3;
    break;

    case 3: if(Dest_X == X) Dir = 2;
    else Dir = 0;
    break;

    case 4: Dir = 1;
    break;

    default: cerr << "dir prob" << ' ' << Color << ' ' << X
              << ' ' << Y << ' ' << NextTime.N <<endl;
    }
    return true;
}

bool Veh::Set_EnDir(long ED)
{
    EnDir = ED;
    return true;
}

bool Veh::Chg_Pwr(double p)
{
    Power = Power - p;
    return true;
}

void Veh::Set_Obj()                    //sets obj locn based on
{                                       //tank's location in formation
    Veh* RB = NULL;
    long ctr = 0;
    long OffY, GrX, GrY;
    GrX = GrY = 0;

    RB = this;                          //set ptr to run thru friendlies
    while(RB != NULL)                   //find center of formation
    {
        GrX += RB->X;
        GrY += RB->Y;
    }
}

```

```
        RB = RB->Next;
        ctr++;
    }

    OffY = GrY/ctr - Y;

    ObjX = XObj;                               //Apply offset to find ind obj
    ObjY = YObj - OffY;
}
```

```

////////////////////////////////////
//                                                                    //
// veh.h:          Header file for the Vehicle class.          //
//                                                                    //
////////////////////////////////////

#ifndef VEH_H
#define VEH_H

#define STRICT

struct TGT {
    long X;                //coord. of tgt or move
    long Y;
    double D;
    TGT* Next;            //points to next tgt in list
};

struct ArTGT {
    long X;                //coord. of tgt or move
    long Y;
    double D;
    long pri;
    double time;
    ArTGT* Next;          //points to next arty tgt in list
};

class Veh
{
public:

    long X;                //current location
    long Y;
    long Dest_X;           //where tank is going
    long Dest_Y;
    long Last_X;
    long Last_Y;
    long ObjX;             //coordinates of objective
    long ObjY;
    long Dir;              //Direction veh is facing (0-7,
                            // 0=N, 4=S)

    //physical genes

```



```

long Vision;           //Vision distance (grids)
double pd;             //prob of det
double w;              //width of veh (m)
double l;              //length of the veh (m)
double h;              //height of the veh (m)
double Move_Rate;     //speed across environment
long Type;            //type of veh (0:bl tk, 1:bl arty,
                        //          2:red tk, 3:red arty)

double Power;         //current effectiveness of veh
char Color;           //veh color (r or b)
long G_Rg;            //Rg of Main Gun in grids
double acc;           //accuracy of gun/ammo comb
double stacc;         //accuracy of sight
double Armor;         //thickness of frontal armor
double Pen;           //Penetration of bullet
bool Moving;          //Flag if moving
long State;           //State of the Tank
long Rds;             //number of rounds on board
long enctr;           //counts en in sight
long frctr;           //counts fr in sight
long arctr;           //counts fr arty in sight
long EnDir;           //general dir of enemy formation
double CFF;           //probability of calling for
                        //          artillery fire

double Reload;        //Time after a shot that veh
                        //          can fire again
long Fix;             //0 if veh is part of fixing
                        //          (shooting) force

struct {              //event list
    double N;         //next event time
    double Sh;        //time of next shot
    double Mv;        //time of next movement
} NextTime;

Veh* Next;            //ptr to next veh in ord'd linked
                        //          list
TGT* Fr;              //pointers to tgt lists and
                        //          closest arty piece

TGT* Arty;
TGT* En;
ArTGT* Target;
long tgtctr;
long Width;           //sheaf width for arty

```

```
long Shots_Msn;                //number of shots/msn

//tactical genes

long Nof;
long Noa;
long Noe;
long Noo;

long ALoF;
long ALoa;
long ALoe;
long ALoo;

long AMedf;
long AMeda;
long AMede;
long AMedo;

long AHif;
long AHia;
long AHie;
long AHio;

long APanf;
long APana;
long APane;
long APano;

long DLoF;
long DLoa;
long DLoe;
long DLoo;

long DMedf;
long DMeda;
long DMede;
long DMedo;

long DHif;
long DHia;
long DHie;
long DHio;

long DPanf;
```

```

long DPana;
long DPane;
long DPano;

long O_Dist;                                //dist to friendlies

Veh (long x, long y, char clr, char arr[], int n);
virtual ~Veh ();

virtual bool Choose_Next_Move(long f, long e, long o); //find next move
virtual double Fr_Locn(long frX, long frY, long &fx, long &fy);
//locate best move based on:
//fr's, en, obj
virtual double En_Locn(long enX, long enY, long &ex, long &ey);
virtual double Obj_Locn(long obX, long obY, long &ox, long &oy);

bool   Chg_Pwr(double p);                    //chgs power rating due to wounding
bool   MoveTo(long X, long Y);               //places tk at new locn
bool   PutFr(long fr);                       //chgs num of fr & en
bool   PutEn(long en);
bool   PutAr(long ar);
double Ratio();                              //computes local force ratio
bool   SetState(long x);                     //sets state of tk
virtual bool ClearPtrs();                    //cleans fr & en tgts from Fr
//and En ptrs
bool   Set_Dir(long D);                      //chgs dir flag
bool   Set_EnDir(long ED);                  //records dir enemy is facing
void   Set_Obj();                            //sets obj locn for tank
};

#endif

```

```

////////////////////////////////////
//                                                                    //
// Vehmgr.cpp: Controls the armies.                                    //
//                                                                    //
////////////////////////////////////

#define STRICT
#define sqr(x)    ((x) * (x))
#define pi       3.1415926

#include "vehmgr.h"
#include "veh.h"
#include "tank.h"
#include "art.h"
#include "BTank.h"
#include "BArt.h"
#include "rvgs.h"
#include "rngs.h"
#include "Param.h"
#include "RParam.h"
#include "land.h"
#include <cmath>
#include <iostream>
#include <string>

using namespace std;

extern World Cell[Landscape_Size][Landscape_Size];
extern double Cur_Time;
extern long BNum;
extern long BAnum;

extern long RTNum;
extern long RANum;

extern long BNum;
extern long RNum;

extern double N[];

long   BEngine;           //selects type engine
long   BAmmo_Type;       //type ammo (and gun)
long   BAmmo_Qty;        //how many stowed rounds/5
long   BSight;           //select type sight

```

```

long   BAuto;                //0=no autoloader, 1=autoloader
long   BArmor;              //amount of armor(*.1m)
long   Dollar_Split;        //Amount of money that goes to tanks

long   BAEngine;           //selects type engine
long   BAmmo_Type;         //type ammo (and gun)
long   BAmmo_Qty;          //how many stowed rounds/5
long   BAAuto;             //0=no autoloader, 1=autoloader
long   BAArmor;            //amount of armor(*.1m)

enum State{No, ALo, AMed, AHi, APanic, DLo, DMed, DHi, DPanic};
//define State variables

int Conv(char arr[], int n, int n1, int n2)
{
    string str1(arr+n1, arr+n2);
    if(str1 == "0" || str1 == "00" || str1 == "000" || str1 == "0000" )
        return 0;
    else if(str1 == "1" || str1 == "01" || str1 == "001" || str1 == "0001")
        return 1;
    else if(str1 == "10" || str1 == "010" || str1 == "0010") return 2;
    else if(str1 == "11" || str1 == "011" || str1 == "0011") return 3;
    else if(str1 == "100" || str1 == "0100") return 4;
    else if(str1 == "101" || str1 == "0101") return 5;
    else if(str1 == "110" || str1 == "0110") return 6;
    else if(str1 == "111" || str1 == "0111") return 7;
    else if(str1 == "1000") return 8;
    else if(str1 == "1001") return 9;
    else if(str1 == "1010") return 10;
    else if(str1 == "1011") return 11;
    else if(str1 == "1100") return 12;
    else if(str1 == "1101") return 13;
    else if(str1 == "1110") return 14;
    else if(str1 == "1111") return 15;
    cerr << "prob in vehmgr.cpp line 65" << endl;
    return 0;
}

Veh* Init_BT(char arr[], int n)
{
    Veh* Tk = NULL;
    long i, j, k, BTStart, max;
    char b = 'b';
    double amtarmor, cost;
    BEngine = Conv(arr, n, 0, 2);

```

```

BAmmo_Type = Conv(arr, n, 2, 6);
BAmmo_Qty  = Conv(arr, n, 6, 10);
BSight     = Conv(arr, n, 10, 12);
BAuto      = Conv(arr, n, 12, 13);
BArmor     = Conv(arr, n, 13, 17);
Dollar_Split = Conv(arr, n, 105, 108);

amtarmor = 2*(2.8-(BAuto*0.6))*2.15*(BArmor*0.1); //m^3 of armor on tank

//compute cost
cost = Engine[BEngine][3] + (5*BAmmo_Qty*Ammo[BAmmo_Type][3]) +
      Ammo[BAmmo_Type][8] + Sight[BSight][3] + (BAuto*Autoloader[2]) +
      (amtarmor*Arm_Cost);

BTStart = long(250000*Dollar_Split*0.1428/cost);

if(BTStart < 1) return Tk;
long BArr[BTStart*4];

if(BTStart*4 < Landscape_Size)
{
    max = BTStart*4;
    for(i=0;i<max;i++) BArr[i] = (Landscape_Size/2)-(2*BTStart)+i;
}
else if(BTStart*2 < Landscape_Size)
{
    max = BTStart*2;
    for(i=0;i<max;i++) BArr[i] = (Landscape_Size/2)-BTStart+i;
}
else
{
    max = Landscape_Size;
    for(i=0;i<max;i++) BArr[i] = i;
}

j = max - 1;
k = Equilikely(0, j);

Tk = new Tank(Blue_Tank_Start, BArr[k], b, arr, n);
//Initializes the first blue tank

BNum++;
BTNum++;
BArr[k] = BArr[j];
j--;

```

```

Veh* Temp;
Temp = Tk;

for(i=1; i<BTStart; i++)          //then the rest of the tribe
{
    k = Equilikely(0, j);
    Temp->Next = new Tank(Blue_Tank_Start, BArr[k], b, arr, n);
    BArr[k] = BArr[j];
    j--;
    Temp = Temp->Next;
    BNum++;
    BNum++;
}
Temp->Next = NULL;
return Tk;
}

Veh* Init_BA(char arr[], int n)
{
    Veh* Arty = NULL;
    long i, j, k, BStart;
    char b = 'b';
    double amtarmor, cost;

    BAEngine      = Conv(arr, n, 108, 110);
    BA Ammo_Type  = Conv(arr, n, 110, 112);
    BA Ammo_Qty   = Conv(arr, n, 112, 116);
    BAAuto        = Conv(arr, n, 241, 242);
    BAArmor       = Conv(arr, n, 116, 119);

    amtarmor=(BAArmor*0.02)*((AAmmo[BA Ammo_Type][7]*AAmmo[BA Ammo_Type][8])+
                             (AAmmo[BA Ammo_Type][8]*AAmmo[BA Ammo_Type][9])+
                             (AAmmo[BA Ammo_Type][7]*AAmmo[BA Ammo_Type][9]));

    //compute cost
    cost = AEngine[BAEngine][3] + (5*BA Ammo_Qty*AAmmo[BA Ammo_Type][3]) +
          AAmmo[BA Ammo_Type][10] + (BAAuto*AAutoloader[1]) + (amtarmor*Arm_Cost);

    BStart = long(250000*(1-(Dollar_Split*0.1428))/cost);
                                     //Starting number of blue arty

    long max;
    long BArr[Landscape_Size];

    if(BStart < 1) return Arty;
    if(BStart*4 < Landscape_Size)

```

```

    {
        max = BASTart*4;
        for(i=0;i<max;i++) BArr[i] = (Landscape_Size/2)-(2*BASTart)+i;
    }
else if(BASTart*2 < Landscape_Size)
    {
        max = BASTart*2;
        for(i=0;i<max;i++) BArr[i] = (Landscape_Size/2)-BASTart+i;
    }
else
    {
        max = Landscape_Size;
        for(i=0;i<max;i++) BArr[i] = i;
    }

j = max - 1;
k = Equilikely(0, j);

Arty = new Art(Blue_Arty_Start, BArr[k], b, arr, n);
//Initializes the first blue arty

BNum++;
BANum++;
BArr[k] = BArr[j];
j--;

Veh* Temp;
Temp = Arty;

for(i=1; i<BASTart; i++) //then the rest of the tribe
    {
        k = Equilikely(0, j);
        Temp->Next = new Art(Blue_Arty_Start, BArr[k], b, arr, n);
        BArr[k] = BArr[j];
        j--;
        Temp = Temp->Next;
        BNum++;
        BANum++;
    }
Temp->Next = NULL;
return Arty;
}

Veh* Init_RT(char arr[], int n)

```



```

{
    Veh* Tk = NULL;
    long i, j, k;
    char r = 'r';
    long RArr[RTStart*4];
    for(i=0;i<RTStart*4;i++) RArr[i] = (Landscape_Size/2)-(2*RTStart)+i;
    j = (RTStart*4) - 1;
    k = Equilikely(0, j);

    Tk = new Tank(Red_Tank_Start, RArr[k], r, arr, n);
                                     //Initializes the first red tank

    RNum++;
    RTNum++;
    RArr[k] = RArr[j];
    j--;

    Veh* Temp;
    Temp = Tk;

    for(i=1; i<RTStart; i++)           //then the rest of the tribe
    {
        k = Equilikely(0, j);
        Temp->Next = new Tank(Red_Tank_Start, RArr[k], r, arr, n);
        RArr[k] = RArr[j];
        j--;
        Temp = Temp->Next;
        RNum++;
        RTNum++;
    }
    Temp->Next = NULL;
    return Tk;
}

Veh* Init_RA(char arr[], int n)
{
    Veh* Arty = NULL;
    long i, j, k;
    char r = 'r';
    long RArr[RASStart*4];
    for(i=0;i<RASStart*4;i++) RArr[i] = (Landscape_Size/2)-(2*RASStart)+i;
    j = (RASStart*4) - 1;
    k = Equilikely(0, j);

    Arty = new Art(Red_Arty_Start, RArr[k], r, arr, n);
                                     //Initializes the first red arty

```

```

RNum++;
RANum++;
RArr[k] = RArr[j];
j--;

Veh* Temp;
Temp = Arty;

for(i=1; i<RAStart; i++)           //then the rest of the tribe
{
    k = Equilikely(0, j);
    Temp->Next = new Art(Red_Arty_Start, RArr[k], r, arr, n);
    RArr[k] = RArr[j];
    j--;
    Temp = Temp->Next;
    RNum++;
    RANum++;
}
Temp->Next = NULL;
return Arty;
}

Veh* ProcTk(Veh* V)
{
    if(V->NextTime.N == V->NextTime.Sh)           //next event is a shot
    {
        ShootTk(V);
        V = Put_In_Order(V);
    }
    else                                           //else event is a mvmt
    {
        Arr(V);
        Chk_State(V);
        ActTk(V);
        V = Put_In_Order(V);
    }
    return V;
}

Veh* ProcArt(Veh* V)
{
    if(V->NextTime.N == V->NextTime.Sh)           //next event is a shot
    {
        ShootArt(V);
        V = Put_In_Order(V);
    }
}

```

```

    }
else //else event is a mvmt
    {
        Arr(V);
        Chk_State(V);
        ActArt(V);
        V = Put_In_Order(V);
    }
return V;
}

ArTGT* ProcImp(ArTGT* Imp)
{
    double p;
    ArTGT* T1 = Imp;
    Imp = Imp->Next;
    Veh* T2 = Cell[T1->X][T1->Y].Occ;

    if(T2 != NULL) //a target is at grid
        {
p = Uniform(0,1);

if(p < (T2->l)*2.15/625.0) T2 = Wnd(T2, p); //round struck vehicle
        }
    T2 = NULL;
    delete T2;
    T1->Next = NULL;
    delete T1;
    T1 = NULL;
    return Imp;
}

Veh* Chk_State(Veh* V)
{
    double p = Uniform(0,1);

    V = Look(V); //look for en and fr vehicles

    if( (p <= V->CFF) && (V->enctr > 0) ) Call_Fire(V);
    //if en visible attempt to CFF

    V->Set_Obj(); //set the objective

    V = Set_State(V); //set the atk/def state of vehicle
}

```

```

    return V;
}

void ActTk(Veh* Tk)
{
    switch ( State(Tk->State) )           //cast long as a State variable
    {
        case No :   Tk->NextTime.Sh = Inf; //ensure tk does not shoot
                   Tk->Choose_Next_Move(Tk->Nof, Tk->Noe, Tk->Noo);
                                           //Choose best move for veh
                   break;

        case ALo :  Tk->Choose_Next_Move(Tk->ALof, Tk->ALoe, Tk->ALoo);
                   break;

        case AMed : Tk->Choose_Next_Move(Tk->AMedf, Tk->AMede, Tk->AMedo);
                   break;

        case AHi :  Tk->Choose_Next_Move(Tk->AHif, Tk->AHie, Tk->AHio);
                   break;

        case APanic : Tk->Choose_Next_Move(Tk->APanf, Tk->APane, Tk->APano);
                   break;

        case DLo :  Tk->Choose_Next_Move(Tk->DLof, Tk->DLoe, Tk->DLo);
                   break;

        case DMed : Tk->Choose_Next_Move(Tk->DMedf, Tk->DMede, Tk->DMedo);
                   break;

        case DHi :  Tk->Choose_Next_Move(Tk->DHif, Tk->DHie, Tk->DHio);
                   break;

        default: Tk->Choose_Next_Move(Tk->DPanf, Tk->DPane, Tk->DPano);
    }

    if(Tk->NextTime.Mv < Tk->NextTime.Sh) Tk->NextTime.N = Tk->NextTime.Mv;
    else Tk->NextTime.N = Tk->NextTime.Sh;
}

void ActArt(Veh* V)
{
    Art* W = (Art*) V;
    if(V->Color == 'b')           //arty is blue

```

```

    {
        switch ( State(W->State) )      //cast long as a State variable
    {
        //Choose best move for arty based on threat
    case No :   W->Choose_Next_Move(W->Nof, W->Noa, W->Noe, W->Noo);
        break;

    case ALo :  W->Choose_Next_Move(W->ALof, W->ALoa, W->ALoe, W->ALoo);
        break;

    case AMed : W->Choose_Next_Move(W->AMedf, W->AMeda, W->AMede, W->AMedo);
        break;

    case AHi :  W->Choose_Next_Move(W->AHif, W->AHia, W->AHie, W->AHio);
        break;

    case APanic : W->Choose_Next_Move(W->APanf, W->APana, W->APane, W->APano);
        break;

    case DLo :  W->Choose_Next_Move(W->DLof, W->DLoa, W->DLoe, W->DLoo);
        break;

    case DMed : W->Choose_Next_Move(W->DMedf, W->DMeda, W->DMede, W->DMedo);
        break;

    case DHi :  W->Choose_Next_Move(W->DHif, W->DHia, W->DHie, W->DHio);
        break;

    default: W->Choose_Next_Move(W->DPanf, W->DPana, W->DPane, W->DPano);
    }
    }

    else                                     //arty is red
    {
        switch ( State(W->State) )      //cast long as a State variable
    {
    case No :   W->Choose_Next_Move(RANof, RANoa, RANoe, RANoo);
        //Choose best move for arty based on threat
        break;

    case ALo :  W->Choose_Next_Move(RAALof, RAALoa, RAALoe, RAALoo);
        break;

    case AMed : W->Choose_Next_Move(RAAMedf, RAAMeda, RAAMede, RAAMedo);
        break;
    }
    }

```

```

case AHi : W->Choose_Next_Move(RAAHif, RAAHia, RAAHie, RAAHio);
    break;

case APanic : W->Choose_Next_Move(RAAPanf, RAAPana, RAAPane, RAAPano);
    break;

case DLo : W->Choose_Next_Move(RADLof, RADLoa, RADLoe, RADLoo);
    break;

case DMed : W->Choose_Next_Move(RADMedf, RADMeda, RADMede, RADMedo);
    break;

case DHi : W->Choose_Next_Move(RADHif, RADHia, RADHie, RADHio);
    break;

default: W->Choose_Next_Move(RADPanf, RADPana, RADPane, RADPano);
}
}

if(V->NextTime.Mv < V->NextTime.Sh) V->NextTime.N = V->NextTime.Mv;
else V->NextTime.N = V->NextTime.Sh;
}

bool Arr(Veh* Tk)
{
    Veh* T1;
    long x, y, destx, desty;
    double p;
    x = Tk->X;
    y = Tk->Y;
    destx = Tk->Dest_X;
    desty = Tk->Dest_Y;

    if((x != destx)|| (y != desty)) //only move if going to
        // a new destination
    {
        //confirm that dest is empty, if not, choose an adjoining grid
        while(Cell[destx][desty].Color != 'u') //dest is occupied
    {
        p = Uniform(0,1);
        if(p > 0.5) destx++;
        else destx--;
        p = Uniform(0,1);
        if(p > 0.5) desty++;
        else desty--;
    }
}

```

```

    if(destx < 0 || destx >= Landscape_Size || desty < 0 ||
        desty >= Landscape_Size)
        //selected location is off grid, start back at self
        {
            destx = x;
            desty = y;
        }
}

//chg number on each point
Cell[x][y].Number_On_Point--;
Cell[destx][desty].Number_On_Point++;

//chg old data and pointer

    Cell[x][y].Occ = NULL;
    Cell[x][y].Color = 'u';
    Cell[x][y].Type = 5;

//chg dest data and pointer

    Cell[destx][desty].Occ = Tk;
    Cell[destx][desty].Type = Tk->Type;
    Cell[destx][desty].Color = Tk->Color;

    T1 = NULL;
    Tk->MoveTo(destx, desty);
}
return true;
}

Veh* Look(Veh* Tk)
{
    extern Veh* BT;
    extern Veh* BA;
    extern Veh* RT;
    extern Veh* RA;

    long X, Y, vis, enctr, frctr, arctr, a, b, i, ED, Best;
    double dist, prob, p, pd;
    long EnDir[8]; //counts number of enemy facing a direction
    TGT* Temp = NULL;
    TGT* T = NULL;
    TGT* T2 = NULL;
    Veh* RB = NULL;

```

```

enctr = 0; //count enemy vehicles in vis area
frctr = 1; //count fr's (self = 1)
arctr = 0;
Tk->ClearPtrs(); //remove fr and en tgts from ptrs
X = Tk->X;
Y = Tk->Y;
vis = Tk->Vision;
pd = Tk->pd;
for(i=0; i<8; i++) EnDir[i] = 0; //initializes array

// build enemy target list

if(Tk->Color== 'b') RB = RT; //look at enemy tanks first
else RB = BT;

while(RB != NULL)
{
  a = RB->X; //get x and y coords for en tank
  b = RB->Y;
  dist = Dist(X, Y, a, b); //compute dist
  if(dist <= vis) prob = pd * sqrt( sqrt(1.0-dist/vis) );
  //inside vision range
  else prob = 0.0; //outside vis range
  p = Uniform(0.0,1.0);

  if(p <= prob) //inside vision range
  {
    enctr++; //increment enemy ctr
    EnDir[RB->Dir]++; //increment enemy direction
    Temp = new TGT;
    Temp->X = a; //get x and y coords for en tank
    Temp->Y = b;
    Temp->D = dist; //dist to en
    Temp->Next = NULL;
    if( Tk->En == NULL) Tk->En = Temp;
    else
    {
      T = Tk->En;
      if(Temp->D < T->D) //if new en is closer, keep new
      {
        Tk->En = Temp;
        T->Next = NULL;
        delete T;
        T = NULL;
      }
    }
  }
}

```



```

    }
}
    RB = RB->Next;
}

if(Tk->Color== 'b') RB = RA;           //then look at en arty
else RB = BA;

while(RB != NULL)
{
    a = RB->X;           //get x and y coords for en arty
    b = RB->Y;
    dist = Dist(X, Y, a, b);           //compute dist
    if(dist <= vis)           //inside vision range
    {
        enctr++;           //increment enemy ctr
        Temp = new TGT;
        Temp->X = a;           //get x and y coords for arty
        Temp->Y = b;
        Temp->D = dist;           //dist from fr to en
        Temp->Next = NULL;
        if( Tk->En == NULL) Tk->En = Temp;
        else
        {
            T = Tk->En;
            if(Temp->D < T->D)           //if new en is closer, keep new
            {
                Tk->En = Temp;
                T->Next = NULL;
                delete T;
                T = NULL;
            }
        }
    }
    RB = RB->Next;
}

if(enctr > 0)           //do if enemy are visible
{
    ED = 0;
    Best = EnDir[0];           //find general direction enemy faces
    for(i=0; i<8; i++)
    {
        if(EnDir[i] > Best)
        {

```

```

        Best = EnDir[i];
        ED = i;
    }
}
    Tk->Set_EnDir(ED);
}

//    build friendly target list

if(Tk->Color== 'r') RB = RT;           //tanks first again
else RB = BT;

while(RB != NULL)
{
    a = RB->X;                          //get x and y coords for fr tank
    b = RB->Y;
    dist = Dist(X, Y, a, b);
    if(dist <= vis && dist > 0.0)      //inside vision range but not self
{
    Temp = new TGT;
    Temp->X = a;                        //record x and y for fr
    Temp->Y = b;
    Temp->D = dist;                     //dist to fr
    Temp->Next = NULL;
    if( frctr == 1)                    //first fr found
    {
        Tk->Fr = Temp;
        Temp = NULL;
    }
    else if ( frctr == 2 )             //second fr found
    {
        T = Tk->Fr;
        if(T->D < Temp->D)
{
        T->Next = Temp;
        Temp = NULL;
}
        else
{
        Tk->Fr = Temp;
        Temp->Next = T;
}
        T = NULL;
        Temp = NULL;
    }
}
}

```

```

else if(frctr > 2)                //mult fr's fd
{
    T = Tk->Fr;
    T2 = T->Next;
    if(Temp->D >= T2->D)
{
    Temp->Next = NULL;
    delete Temp; //new fr is 3d closest, goodbye
    Temp = NULL;
}
    else if(Temp->D >= T->D)        //second closest
{
    T->Next = Temp;
    T2->Next = NULL;
    delete T2;
    T2 = NULL;
}
    else                            //closest
{
    Tk->Fr = Temp;
    Temp->Next = T;
    T->Next = NULL;
    T2->Next = NULL;
    delete T2;
    T2 = NULL;
}
    Temp = NULL;
    T = NULL;
    T2 = NULL;
}
frctr++;
}
    RB = RB->Next;
}

if(Tk->Color== 'r') RB = RA;        //do fr arty
else RB = BA;

while(RB != NULL)
{
    a = RB->X;                        //get x and y coords for fr arty
    b = RB->Y;
    dist = Dist(X, Y, a, b);
    if(dist <= vis && dist > 0.0) //inside vision range but not self
{

```

```

    arctr++;                                //increment arctr
    Temp = new TGT;
    Temp->X = a;                             //get x and y coords for fr
    Temp->Y = b;
    Temp->D = dist;                           //dist to fr
    Temp->Next = NULL;
    if( arctr == 1) Tk->Arty = Temp; //first fr found

    else                                     //mult fr found
    {
        T = Tk->Arty;
        if(T->D <= Temp->D)
    {
        Temp->Next = NULL;
        delete Temp; //new arty is farther than prev. found
        Temp = NULL;
    }
        else
    {
        Tk->Arty = Temp;
        T->Next = NULL;
        delete T;
        T = NULL;
    }
        }
    T = NULL;
    Temp = NULL;
}
    RB = RB->Next;
}

    Tk->PutFr(frctr);                        //store number of frdly, arty, and en
    Tk->PutAr(arctr);
    Tk->PutEn(enctr);
    return Tk;
}

```

```

Veh* Set_State(Veh* Tk)
{
    TGT* Temp = Tk->En;
    double ratio, p;
    ratio = Tk->Ratio();
    if(ratio == 0.0) Tk->SetState(No); //State = No
    else if(ratio < 0.1 && Temp->D > 80) Tk->SetState(ALo); //ALo
}

```

```

else if(ratio < 0.3 && Temp->D > 40)    Tk->SetState(AMed); //AMed
else if(ratio < 0.5)                    //could go to AHi or APanic
{
    p = Uniform(0,1);
    if(p < 0.90) Tk->SetState(AHi);      //p=0.90 of ->AHi
    else Tk->SetState(APanic);          //go to APanic
}
else if(ratio < 1.0 && Temp->D > 80)    Tk->SetState(DLo); //State = DLo
else if(ratio < 5.0 && Temp->D > 40)    Tk->SetState(DMed); //State = DMed
else
{
    p = Uniform(0,1);
    if(p < 0.90) Tk->SetState(DHi);
    else Tk->SetState(DPanic);          //go to DPanic
}
return Tk;
}

void ShootTk(Veh* Tk)
{
    double p, dist, ph, pk, a;
    static long rshot = 0;
    static long bshot = 0;
    Veh* T = NULL;                //Tgt'd tank
    TGT* Tgt = NULL;              //ptr to struc TGT to id tgt'd tank
    long SD, VD;                  //Shooter Direction and Victim Direction

    Tgt = Tk->En;
    T = Cell[Tgt->X][Tgt->Y].Occ;
    if(T != NULL)
    {
        dist = Tgt->D;                //distance to tgt
        if(dist < Tk->G_Rg)            //tgt is in range
        {
            if(Tk->Color == 'r') rshot++; //count shots on each side
            else bshot++;
        }

        ph = T->w*40/dist;              //apparent width of target in mils

        //adjust for moving and stationary firers and targets
        //stationary tgts get ph cut to 1/3
        //mvg shooting at mvg reduces accuracy by 10%

        if(Tk->Moving == false && T->Moving == false) ph = 0.33*ph; //S->S
        else if(Tk->Moving == true && T->Moving == false) ph = 0.225*ph; //M->S
    }
}

```

```

else if(Tk->Moving == true && T->Moving == true) ph = 0.9*ph;    //M->M

ph = ph/2.0;                //half of apparent width in mils
p = Normal(0.0, Tk->acc) + Normal(0.0, Tk->stacc);
                                //acc of gun/ammo and sight
if(p < 0.0) p = -p;

if(p <= ph)                    //tgt is hit
{
    if(T->Pen>Tk->Armor)
{
    a = asin(T->Pen/Tk->Armor);
    pk = (pi-a)/pi;
}
    else pk = 0.0;

    //adjust for frontal or flank shots
    SD = Tk->Dir;
    VD = T->Dir;
    if((SD+4)%8 == VD-1 || (SD+4)%8 == VD || (SD+4)%8 == VD+1) pk = pk/2;
    p = Uniform(0,1);          //draw to see if killed or wound
    if(p < pk )
{
    T = Kill(T);                //tgt killed
    Tk->NextTime.Sh = Inf;      //do not shoot again at tgt
}
    else Wnd(T, 1-p);          //tgt wounded
    Tk->NextTime.Sh += Tk->Reload; //shoot again after reloading
}
} //if(dist < Tk->G_Rg)
    else {                    //tgt is out of rg, don't sched next shot
Tk->NextTime.Sh = Inf;
    }
}

else {                        //tgt is gone, don't sched another shot
    Tk->NextTime.Sh = Inf;
}

if(Tk->NextTime.Mv < Tk->NextTime.Sh) Tk->NextTime.N = Tk->NextTime.Mv;
else Tk->NextTime.N = Tk->NextTime.Sh;
}

void ShootArt(Veh* Arty)
{

```

```

extern ArTGT* Imp;
extern double Cur_Time;

ArTGT* T1 = Arty->Target;
ArTGT* T2 = NULL;
long x, y, y1, y2, i;
double TOF;
//dequeue target and process mission
Arty->Target = T1->Next;
Arty->tgtctr--;
if(Arty->Target == NULL)
{
    if(Arty->tgtctr != 0) cerr << "prob in shootart, time " <<
        Cur_Time << endl;
    Arty->NextTime.Sh = Inf;
    Arty->NextTime.N = Arty->NextTime.Mv;
}
else
{
    Arty->NextTime.Sh += Arty->Reload;
    if(Arty->NextTime.Sh < Arty->NextTime.Mv) Arty->NextTime.N =
        Arty->NextTime.Sh;
    else Arty->NextTime.N = Arty->NextTime.Mv;
}

//schedule impact of rounds
for(i=0; i<Arty->Shots_Msn; i++)
{
    x = T1->X;
    y1 = max( 0, T1->Y - long(Arty->Width/2.0) );
    y2 = min( T1->Y + long(Arty->Width/2.0), Landscape_Size );
    y = Equilikely(y1, y2);

    T2 = new ArTGT;
    T2->X = x;
    T2->Y = y;
    T2->D = Dist(x, y, Arty->X, Arty->Y);
    TOF = double(T2->D)/double(Arty->G_Rg);
    T2->time = Cur_Time + TOF;
    T2->pri = 0;
    T2->Next = Imp;
    Imp = T2;
    Imp = OrderTgts(Imp);
    T2 = NULL;
}

```

```

T1 = NULL;
if(Imp != NULL) N[4] = Imp->time;
}

ArTGT* OrderTgts(ArTGT* Imp)
{
  ArTGT* T1 = Imp;
  ArTGT* T2 = Imp->Next;
  ArTGT* T3 = Imp;

  if(T2 == NULL)          //there is only one target, so return
  {
    T1 = NULL;
    T3 = NULL;
    return Imp;
  }

  if(Imp->time > T2->time) //new Imp is later than most imminent
  {
    Imp = Imp->Next;
    while( T1->time > T2->time && T2->Next != NULL)
      //find proper location for new Imp
    {
      T3 = T2;
      T2 = T2->Next;
    }
    if(T2->time >= T1->time)
    {
      T1->Next = T2;
      T3->Next = T1;
    }
    else
    {
      T2->Next = T1;
      T1->Next = NULL;
    }
  }
  T1 = NULL;
  T2 = NULL;
  T3 = NULL;
  return Imp;
}

Veh* Wnd(Veh* Tk, double p)
{

```



```

    if(p > Tk->Power) Tk = Kill(Tk);    //if p > pwr, tgt is killed
    else Tk->Chg_Pwr(p);                //reduce pwr by p
    return Tk;
}

Veh* Kill(Veh* V)
{
    extern Veh* BT;
    extern Veh* BA;
    extern Veh* RT;
    extern Veh* RA;

    Veh* T1 = NULL;
    Veh* T2 = NULL;

    int x, y;

    switch(V->Type)
    {
        case 0 : T1 = BT;
                break;

        case 1 : T1 = BA;
                break;

        case 2 : T1 = RT;
                break;

        case 3 : T1 = RA;
                break;

        default : cerr << " problem in kill " << endl;
    }

    x = V->X;
    y = V->Y;

    if(T1 == V)                        //Veh to be killed is first in queue
    {
        switch(V->Type)
        {
            case 0 : BT = T1->Next;
                    break;

```

```

case 1 : BA = T1->Next;
        break;

case 2 : RT = T1->Next;
        break;

case 3 : RA = T1->Next;
        break;

default : cerr << " problem in kill 2 " << endl;
}
    T1->ClearPtrs();
    T1->Next = NULL;
    delete V;
    V = NULL;
    T1 = NULL;
}
else
{
    T2 = T1;                //increment thru list to find victim
    T1 = T1->Next;
    while(T1 != V)
{
    T2 = T1;
    T1 = T1->Next;
}
    T2->Next = T1->Next;
    T1->Next = NULL;
    delete V;                //destroy victim tk
    T1 = NULL;
    T2 = NULL;

}
clearCell(x, y);

return T1;
}

Veh* Put_In_Order(Veh* Tk)
{
    Veh* T1 = Tk;
    Veh* T2 = Tk->Next;
    if(T2 == NULL)            //Tk points to only veh in list
    {
        T1 = NULL;
    }
}

```

```

        return Tk;
    }
    Veh* T3 = Tk;

    if(Tk->NextTime.N > T2->NextTime.N)
    {
        Tk = Tk->Next;
        while((T1->NextTime.N > T2->NextTime.N) && (T2->Next != NULL))
    {
        T3 = T2;
        T2 = T2->Next;
    }
        if(T1->NextTime.N > T2->NextTime.N) //T2->Next == NULL
    {
        T2->Next = T1;
        T1->Next = NULL;
    }
        else //T2->NextTime.N <= T1->NextTime.N
    {
        T3->Next = T1;
        T1->Next = T2;
    }
    }
    T1 = NULL;
    T2 = NULL;
    T3 = NULL;
    return Tk;
}

bool Put_Vehs(Veh* Tk)
{
    long x;
    long y;
    Veh* T1 = Tk;
    while (T1 != NULL)
    {
        x = T1->X;
        y = T1->Y;
        Cell[x][y].Color = T1->Color;
        Cell[x][y].Type = T1->Type;
        Cell[x][y].Number_On_Point++;
        Cell[x][y].Occ = T1;

        T1 = T1->Next;
    }
}

```

```

    T1 = NULL;
    return true;
}

void Call_Fire(Veh* V)
{
    extern Veh* BA;
    extern Veh* RA;

    ArTGT* T1 = NULL;           //tgt that cff wants to hit
    ArTGT* T2 = NULL;
    ArTGT* T3 = NULL;
    TGT* A1 = NULL;           //closest arty to caller
    TGT* T = NULL;
    Veh* A = NULL;           //arty that will shoot mission
    if(V->Color == 'b') A = BA; //if veh is blue, call for fire to blue
    else A = RA;           //arty, else red arty shoots

    if(V->En != NULL && V->Arty != NULL && A != NULL)
        {
            //defensive programming, veh is in
            //contact with tgt and arty and
            //artillery exists

            T1 = new ArTGT;
            T = V->En;
            T1->X = T->X;
            T1->Y = T->Y;
            T1->D = T->D;
            T1->pri = V->State;
            T1->time = Cur_Time;
            T1->Next = NULL;

            A1 = V->Arty;

            while(A1->X != A->X || A1->Y != A->Y) A = A->Next;
            //find closest arty
            A->tgtctr++; //increment the counter

            if( A->Target == NULL )
        {
            A->Target = T1; //if no targets in q, place in q
            A->NextTime.Sh = Cur_Time + A->Reload; //schedule a shot
            if(A->NextTime.N > A->NextTime.Sh) //if next event for this arty is
                //this shot
            {
                A->NextTime.N = A->NextTime.Sh; //update next time
            }
        }
    }
}

```

```

        if(A->Color == 'b') BA = Put_In_Order(BA);
                                //reorder arty based on this shot
        else RA = Put_In_Order(RA);
    }
} //if(A->Target == NULL)
    else //targets exist
{
    T2 = A->Target;
    if(T1->pri > T2->pri) //new target is higher priority than
        { //any in q
            T1->Next = T2;
            A->Target = T1;
        }
    else //higher pri targets exist
        {
            while( (T1->pri <= T2->pri) && T2->Next != NULL )
                //increment thru list
            {
                T3 = T2;
                T2 = T2->Next;
            }

            if(T1->pri <= T2->pri)
            {
                T2->Next = T1;
                T1->Next = NULL;
            }
            else
            {
                T3->Next = T1;
                T1->Next = T2;
            }
            T1 = NULL;
            T2 = NULL;
            T3 = NULL;
        }
    while(A->tgtctr > Max_Msns) //eliminate the last
        { //mission in q
            T2 = A->Target;
            while(T2->Next != NULL)
            {
                T3 = T2;
                T2 = T2->Next;
            }
            A->tgtctr--;
        }
}

```

```
        T3->Next = NULL;
        T2->Next = NULL;
        delete T2;
        T2 = NULL;
    }
} //else targets already exist
}
```

```

////////////////////////////////////
//                                                                    //
// vehmgr.h: Provides the declarations for vehmgr.cpp                //
//                                                                    //
////////////////////////////////////

#include "art.h"
#include "land.h"
#include <string>

#ifndef VEHMGR_H
#define VEHMGR_H

int Conv(char arr[], int n, int n1, int n2);
Veh* Init_BT(char arr[], int n);
Veh* Init_BA(char arr[], int n);

Veh* Init_RT(char arr[], int n);
Veh* Init_RA(char arr[], int n);

bool Put_Vehs(Veh* V);

Veh* ProcTk(Veh* Tk);
Veh* ProcArt(Veh* Arty);
ArTGT* ProcImp(ArTGT* Imp);

bool Arr(Veh* V);

Veh* Chk_State(Veh* V);

void ActTk(Veh* Tk);

void ActArt(Veh* Arty);

Veh* Look(Veh* V);

Veh* Set_State(Veh* V);

void ShootTk(Veh* Tk);

void ShootArt(Veh* Arty);

Veh* Wnd(Veh* V, double p);

Veh* Kill(Veh* V);

```

```
Veh* Put_In_Order(Veh* V);  
ArTGT* OrderTgts(ArTGT* T);  
  
void Call_Fire(Veh* V);  
  
#endif
```



```

/

/////////////////////////////////////////////////////////////////
//                                                                    //
// BTank.h:  Contains the Blue Tank Characteristics                    //
//                                                                    //
/////////////////////////////////////////////////////////////////

#ifndef BTANK_H
#define BTANK_H

double Engine[4][4] =
  { //{wt(mt), hp,      m^3,  cost($k)}
    {1.454, 1500.0, 1.9875, 175.0}, //LV100 Engine
    {3.3,   1500.0, 3.95,   250.0}, //AGT-1500 (Current M1)
    {1.0,   900.0,  2.0,    500.0}, //hybrid
    {3.545, 750.0,  5.035,  100.0}  //AVDS-1790 (M60)
  };

double Ammo[16][10] =
  { //{wt(mt), pen(m), m^3, cost($k), rg(m), accuracy(mils), gun wt,
    gun m^3, gun cost, min/rd
                                     //120mm
    {0.0187, 0.6, 0.272, 1.0, 3000, 1.0, 1.0, 1.5, 250.0, 5}, //M829
    {0.0187, 0.7, 0.272, 2.0, 3000, 0.9, 1.0, 1.5, 250.0, 5}, //M829A1
    {0.0187, 0.8, 0.272, 5.0, 3000, 0.8, 1.0, 1.5, 250.0, 5}, //M829A2
                                     //125mm
    {0.0243, 0.550, 0.294, 2.0, 2500, 1.0, 1.3, 2.0, 350.0, 5}, //BM42M
    {0.0243, 0.60,  0.294, 2.9, 2500, 1.0, 1.3, 2.0, 350.0, 5}, //BK27
    {0.0243, 0.65,  0.294, 2.5, 2500, 0.9, 1.3, 2.0, 350.0, 5}, //BK29
                                     //140mm
    {0.0404, 0.8,   0.369, 4.0, 3000, 0.8, 2.0, 5.0, 500.0, 5}, //
    //ATGM
    //AT-11
    {0.250,  0.8,   0.270, 50.0, 4000, 0.4, 1.0, 1.0, 500.0, 10},
    //TOW2B
    {0.180,  0.8,   0.270, 100.0, 5000, 0.4, 1.0, 1.0, 500.0, 10},
    //FOIT
    {0.350,  1.0,   0.270, 150.0, 5000, 0.3, 1.0, 1.0, 500.0, 10},
    //Javelin
    {0.250,  0.6,   0.200, 75.0, 2000, 0.3, 1.0, 1.0, 500.0, 10},
    //LOSAT
    {0.4,    2.0,   0.403, 250.0, 5000, 0.08, 1.0, 1.0, 500.0, 10},
  };

```

```
//Dragon
{0.150, 0.2, 0.150, 40.0, 1500, 0.3, 1.0, 1.0, 500.0, 10},
//EM Projo
{0.001, 2.0, 0.001, 0.001, 10000, 0.05, 10.0, 4.0, 1250.0, 10},
{0.001, 2.0, 0.001, 0.1, 10000, 0.05, 10.0, 4.0, 1250.0, 10},
{0.001, 2.0, 0.001, 0.001, 10000, 0.05, 10.0, 4.0, 12500.0, 10}
};

double Sight[4][6] =
  { //wt, Pd, m^3, cost($k), accuracy(mils), Rg
    {0.1, 0.4, 0.25, 10.0, 0.8, 2000}, //105D (Daylight only)
    {0.5, 0.6, 0.50, 100.0, 0.8, 2500}, //IR
    {0.75, 0.8, 1.0, 250.0, 0.5, 5000}, //Thermal
    {2.00, 0.95, 4.0, 1000.0, 0.1, 10000} //MMW
  };

double Autoloader[3] = {1.0, -5.0, 150.0}; //wt, m^3, cost($k)

#endif
```

```

/////////////////////////////////////////////////////////////////
//                                                                    //
// Bart.h:  Contains the Blue Arty Characteristics                    //
//                                                                    //
/////////////////////////////////////////////////////////////////

#ifndef BART_H
#define BART_H

double AEngine[4][4] =
  //{wt(mt), hp,    m^3,    cost($k)}
  {3.3,  1500.0, 3.95,  150.0},    //AGT-1500 (Current M1)
  {3.545, 750.0, 5.035, 100.0},    //AVDS-1790 (M60)
  {4.0,   500.0, 5.0,   50.0},     //lighter, cheaper engine
  {2.0,   250.0, 3.0,   45.0}     //M113 engine
};

double AAmmo[4][13] =
  //{wt(mt), pen(m), m^3, cost($k), rg(m), acc(pt or area), gun wt,
  // w, l, h, gun cost, min/rd, max rds/msn
  //M1024 120mm mortar
  {0.022, 0.0025, 0.18, 0.10, 10000.0, 1.0, 6.0, 2.8, 5.15,
   2.3, 500.0, 1.0, 8.0}, //HE
  //M109 155mm howitzer
  {0.031, 0.005, 0.3, 0.20, 18000.0, 1.0, 25.0, 3.1, 6.2,
   3.0, 2500.0, 1.0, 8.0}, //HE
  //Crusader 155mm howitzer
  {0.031, 0.005, 0.3, 0.20, 45000.0, 1.0, 50.0, 3.1, 6.2,
   3.0, 5000.0, 1.0, 8.0}, //HE
  //EFOGM
  {0.001, 0.1, 0.6, 3.0, 40000.0, 0.0, 6.0, 2.8, 5.15,
   2.3, 500.0, 1.0, 1.0} //Ms1
};

double AAutoloader[2] = {1.0, 150.0}; //wt, cost($k)

#endif

```

```

/////////////////////////////////////////////////////////////////
//                                                                    //
// Param.h: Holds the major parameters for the cbt sim.              //
//                                                                    //
/////////////////////////////////////////////////////////////////

#ifndef PARAM_H
#define PARAM_H

#define Landscape_Size      750      //Size of landscape=18,750m
#define Time_Limit          200.0    //Sets the timelimit for the sim
#define POSITION_STREAM      0        //draws random numbers from different
// streams

#define VISION_STREAM       1
#define M_RATE_STREAM      2
#define FIX_FORCE_STREAM   3
#define METAB_STREAM       4
#define Inf                 Time_Limit * 100 // A big number
#define Out                 1        //0 = run visual, 1 = run without
// graphics

#define pixel_size         3

/////////////////////////////////////////////////////////////////
//                                                                    //
// Blue Tank Force Attributes:                                        //
//                                                                    //
/////////////////////////////////////////////////////////////////

#define XObj                Landscape_Size/3 //Objective is in red start area

#define YObj                Landscape_Size/2

#define Arm_Wt              6.4        //Wt of armor per m^3 (mt)

#define Arm_Cost            630.8      //Cost of armor per m^3 ($k)

/////////////////////////////////////////////////////////////////
//                                                                    //
// Blue Tank Attributes:                                          //
//                                                                    //
/////////////////////////////////////////////////////////////////

#define CFF_Min             10        //minutes between calls for arty fire

```

```

////////////////////////////////////
//                                                                    //
// Blue Arty Force Attributes:      //
//                                                                    //
////////////////////////////////////

#define Max_Msns          5      //max # of fire msns in target q

////////////////////////////////////
//                                                                    //
// Blue Arty Attributes:           //
//                                                                    //
////////////////////////////////////

#define A_Rds_Min        1      //num rounds that can be fired/minute

////////////////////////////////////
//                                                                    //
//      Landscape attributes:      //
//                                                                    //
////////////////////////////////////

#define Blue_Tank_Start  5*Landscape_Size/8
                        //Blue will start in lower half
#define Blue_Arty_Start  3*Landscape_Size/4

#endif

```

```

////////////////////////////////////
//                                                                    //
// RParam.h: Holds the major parameters for the red side of //
//           the cbt sim.                                     //
//                                                                    //
////////////////////////////////////

#ifndef RPARAM_H
#define RPARAM_H

#define RTStart      15          //Starting number of red tanks
#define RASStart     5           //Starting number of red arty

////////////////////////////////////
//                                                                    //
//           Red Tank Force Attributes:                          //
//                                                                    //
////////////////////////////////////

#define RFix_Force   0.9        //size of attacker in fixing force

#define ROpt_Dist    200        //Min dist to friendlies

#define RNof         1
#define RNoe         0
#define RNoo         1

#define RALof        1
#define RALoe        5
#define RALoo        1

#define RAMedf       3
#define RAMede       10
#define RAMedo       1

#define RAHif        1
#define RAHie        10
#define RAHio        0

#define RAPanf       0
#define RAPane       10
#define RAPano       0

#define RDLoF        1
#define RDLoe        0

```

```

#define RDLo0    2

#define RDMedf   3
#define RDMede   0
#define RDMedo   2

#define RDHif    1
#define RDHie    1
#define RDHio    1

#define RDPanf   0
#define RDPane   1
#define RDPano   0

/////////////////////////////////////////////////////////////////
//                                                    //
//      Red Tank Attributes:                          //
//                                                    //
/////////////////////////////////////////////////////////////////

#define RHi_Vision    6000 //Hi limit of vision
#define RLo_Vision    5500 //Lo limit of vision

#define RHi_Move_Rate 8000 //Fastest movement allowed (m/hr)
#define RLo_Move_Rate 4000 //Slowest movement allowed

#define RGun_Rg       3000 //Max Range (m) of main gun

#define RRds_Min      1    //num rounds that can be fired/minute

#define RGun_Type     1    //1=conv gun, 2=mssl, 3=em gun
#define RAmmo_Type    1    //1=sabot, 2=heat, 3=he

#define RCFE_Min      10   //minutes between calls for arty fire

/////////////////////////////////////////////////////////////////
//                                                    //
//      Red Arty Force Attributes:                    //
//                                                    //
/////////////////////////////////////////////////////////////////

#define RShoot_Force  0.9  //% of attacker arty force shooting

#define ROpt_A_Dist   200  //Min dist to friendly art

```

```
#define RMax_Msns          5 //max # of msns in target q
#define RShots_Msn        6 //number of shots per fire mission
#define RSheaf_Width      200 //width of a fire mission's sheaf

#define RANof             1
#define RANoa             1
#define RANoe             0
#define RANoo             0

#define RAALof           1
#define RAALoa           1
#define RAALoe           5
#define RAALoo           0

#define RAAMedf          3
#define RAAMeda          3
#define RAAMede          10
#define RAAMedo          0

#define RAAHif           1
#define RAAHia           1
#define RAAHie           10
#define RAAHio           0

#define RAAPanf          0
#define RAAPana          0
#define RAAPane          10
#define RAAPano          0

#define RADLof           1
#define RADLoa           1
#define RADLoe           0
#define RADLoo           0

#define RADMedf          3
#define RADMeda          3
#define RADMede          0
#define RADMedo          0

#define RADHif           1
#define RADHia           1
#define RADHie           1
#define RADHio           0
```



```

#define RADPanf  0
#define RADPana  0
#define RADPane  1
#define RADPano  0

/////////////////////////////////////////////////////////////////
//                                                     //
//      Red Arty Attributes:                          //
//                                                     //
/////////////////////////////////////////////////////////////////

#define RA_Hi_Vis      2000      //Hi limit of vision
#define RA_Lo_Vis      1500      //Lo limit of vision

#define RA_Hi_Move_Rate 5000     //Fastest movement allowed (m/hr)
#define RA_Lo_Move_Rate 4000     //Slowest movement allowed

#define RA_Gun_Rg      15000     //Max Range (m) of main gun

#define RA_Rds_Min     1         //num rounds that can be fired/minute

#define RA_Gun_Type    1         //1=conv gun, 2=msl, 3=em gun
#define RA_Ammo_Type   1         //1=sabot, 2=heat, 3=he

/////////////////////////////////////////////////////////////////
//                                                     //
//      Landscape attributes:                          //
//                                                     //
/////////////////////////////////////////////////////////////////

#define Red_Tank_Start  Landscape_Size/3 //Red will start in upper half
#define Red_Arty_Start  Landscape_Size/4

#endif

```

```

////////////////////////////////////
//                                                                    //
// Tank.cpp : Member functions of the Tank class.                    //
//                                                                    //
////////////////////////////////////

#define STRICT
#include "Param.h"
#include "RParam.h"
#include "tank.h"

extern long BNum;
extern long RNum;
extern double N[5];

////////////////////////////////////
// Constructors, destructors, and overloaded operators:             //
////////////////////////////////////

// default constructor:
Tank::Tank(long x, long y, char clr, char arr[], int n)
        : Veh(x, y, clr, arr, n)
{
}

//tank destructor
Tank::~Tank()
{
    if(Color == 'r') RNum--;
    else BNum--;
}

```

```
////////////////////////////////////  
//                                                                    //  
// tank.h:      Header file for the tank class, derived from //  
//              the veh class                                //  
////////////////////////////////////  
  
#ifndef TANK_H  
#define TANK_H  
#include "veh.h"  
  
#define STRICT  
  
class Tank : public Veh  
{  
private:  
  
public:  
  
    Tank(long x, long y, char clr, char arr[], int n);  
    ~Tank();  
  
};  
  
#endif
```

```

/////////////////////////////////////////////////////////////////
//                                                                 //
// Art.cpp : Member functions of the Artillery class.           //
//                                                                 //
/////////////////////////////////////////////////////////////////

#define STRICT
#include "vehmgr.h"
#include "art.h"
#include "land.h"

using namespace std;

#define sqr(x) ((x)*(x))

extern long BANum;
extern long RANum;

/////////////////////////////////////////////////////////////////
// Constructors, destructors, and overloaded operators:         //
/////////////////////////////////////////////////////////////////

// default constructor:
Art::Art(long x, long y, char clr, char arr[], int n)
                                     : Veh(x, y, clr, arr, n)
{
    extern double AEngine[4][4];
    extern double AAmmo[4][13];
    extern double AAutoloader[2];

    extern long BAEngine;
    extern long BAammo_Type;
    extern long BAammo_Qty;
    extern long BAAuto;
    extern long BAArmor;

    double Shoot_Force = double(Conv(arr, n, 237, 241)) * 0.06666;
                                     //pct of force shooting

    if(clr == 'b') //arty is blue
    {
        double p, wt, spd;
        Vision = 80; //vision is fixed at 2000m
        pd = 0.8;
        w = AAmmo[BAammo_Type][7];
    }
}

```

```

l = AAmmo[BAammo_Type][8];
h = AAmmo[BAammo_Type][9];

//compute wt, armor + ammo + engine&fuel
wt = ( ( (2*(h*w))+(2*(w*1))+(2*(h*1)) )*(BAarmor*0.05)*Arm_Wt ) +
      (AAmmo[BAammo_Type][0]*BAammo_Qty*5) + AEngine[BAEngine][0] +
      BAAuto*AAutoloader[0];

spd = 6.49 + 1.49*AEngine[BAEngine][1]/wt; //converts hp/t to km/hr
SelectStream(M_RATE_STREAM);
Move_Rate = Equilikely(long(0.9*spd),long(1.1*spd))*40.0/60.0;
Type = 1;
G_Rg = long(AAmmo[BAammo_Type][4]/25);
acc = AAmmo[BAammo_Type][5]; //1=area, 0=pt
stacc = 0.0;
Armor = BAarmor*0.05; //thickness of armor
Pen = AAmmo[BAammo_Type][1]; //penetration of ammo
Rds = BAammo_Qty * 5; //number of rounds
// aboard
Reload = AAmmo[BAammo_Type][11]; //time to reload

SelectStream(FIX_FORCE_STREAM);
p = Uniform(0,1);

if(p < Shoot_Force) Fix = 0; //part of shooting force
else Fix = 1;

Width = Conv(arr, n, 126, 129); //width of arty sheaf

//shots per msn is min of genome-driven number and type rd-driven
Shots_Msn = min(Conv(arr,n,122,126),long(AAmmo[BAammo_Type][12]));

//tactical genes
Nof = Conv(arr, n, 129, 132);
Noa = Conv(arr, n, 132, 135);
Noe = Conv(arr, n, 135, 138);
Noo = Conv(arr, n, 138, 141);

ALof = Conv(arr, n, 141, 144);
ALoa = Conv(arr, n, 144, 147);
ALoe = Conv(arr, n, 147, 150);
ALoo = Conv(arr, n, 150, 153);

AMedf = Conv(arr, n, 153, 156);
AMeda = Conv(arr, n, 156, 159);

```

```

    AMede = Conv(arr, n, 159, 162);
    AMedo = Conv(arr, n, 162, 165);

    AHif = Conv(arr, n, 165, 168);
    AHia = Conv(arr, n, 168, 171);
    AHie = Conv(arr, n, 171, 174);
    AHio = Conv(arr, n, 174, 177);

    APanf = Conv(arr, n, 177, 180);
    APana = Conv(arr, n, 180, 183);
    APane = Conv(arr, n, 183, 186);
    APano = Conv(arr, n, 186, 189);

    DLoF = Conv(arr, n, 189, 192);
    DLoa = Conv(arr, n, 192, 195);
    DLoe = Conv(arr, n, 195, 198);
    DLoo = Conv(arr, n, 198, 201);

    DMedf = Conv(arr, n, 201, 204);
    DMeda = Conv(arr, n, 204, 207);
    DMede = Conv(arr, n, 207, 210);
    DMedo = Conv(arr, n, 210, 213);

    DHif = Conv(arr, n, 213, 216);
    DHia = Conv(arr, n, 216, 219);
    DHie = Conv(arr, n, 219, 222);
    DHio = Conv(arr, n, 222, 225);

    DPanf = Conv(arr, n, 225, 228);
    DPana = Conv(arr, n, 228, 231);
    DPane = Conv(arr, n, 231, 234);
    DPano = Conv(arr, n, 234, 237);

    O_Dist = Conv(arr, n, 119, 122);           //opt dist in grids
}

else{                                         //arty is red and attributes hard-coded
    Vision = 80;
    pd = 0.8;
    w = 3.1;
    l = 6.0;
    h = 3.0;
    Move_Rate = Equilikely(15, 18); //+/- 10% of 25kph
    Type = 3;

```

```

G_Rg = 600;           //15000m/25;
acc = 1.0;           //area fire wpn
stacc = 0.0;
Armor = 0.05;
Pen = 0.05;         //50mm pen ability
Rds = 60;           //60 stowed rounds
Reload = 1.0;       //1 min between rounds
Shots_Msn = 6;      //6 shots in a volley
Width = 8;          //width of sheaf = 200m

//tactical genes
Nof = 1;
Noa = 1;
Noe = 0;
Noo = 1;

ALof = 1;
ALoa = 1;
ALoe = 5;
ALoo = 1;

AMedf = 3;
AMeda = 3;
AMede = 10;
AMedo = 1;

AHif = 1;
AHia = 1;
AHie = 10;
AHio = 0;

APanf = 0;
APana = 0;
APane = 1;
APano = 0;

DLoF = 2;
DLoa = 2;
DLoe = 0;
DLoo = 2;

DMedf = 2;
DMeda = 5;
DMede = 0;
DMedo = 1;

```

```

    DHif = 1;
    DHia = 1;
    DHie = 1;
    DHio = 1;

    DPanf = 0;
    DPana = 0;
    DPane = 1;
    DPano = 0;

    O_Dist = 8;                //200m between systems
}

    arctr = 0;
}

//default destructor
Art::~Art()
{
    if(Color == 'r') RANum--;
    else BANum--;
}

bool Art::Choose_Next_Move(long f, long a, long e, long o)
{
    double dist, distf, dista, diste, disto, best;
                                //dist to dest, fr, en, obj and best pri
    long fx, fy, ax, ay, ex, ey, ox, oy, b, i;
    double pri[4];                //holds priority calc

    //find best location based on friendly
    switch (frctr)
    {
        case 1 : fx = X;                //no other fr are in sight
                fy = Y;
                break;

        default: Fr_Locn(Fr->X, Fr->Y, fx, fy);
                                //find locn based on closest fr tk
                break;
    }

    distf = Dist(X, Y, fx, fy);        //find dist and priority of movement
    pri[0] = f * distf;                //based on friendlies

```



```

//find best location based on friendly arty
switch (arctr)
{
case 0 : ax = X;           //no other arty are in sight
        ay = Y;
        break;

default: Ar_Locn(Arty->X, Arty->Y, ax, ay);
        //loc'n of closest arty
        break;
}

dista = Dist(X, Y, ax, ay); //find dist and priority of movement
pri[1] = a * dista;         //based on fr arty

//find best locn based on closest enemy

switch(enctr)
{
case 0 : ex = X;           //no en in sight
        ey = Y;
        break;

default: En_Locn(En->X, En->Y, ex, ey);
}
diste = Dist(X, Y, ex, ey);
pri[2] = e * diste;

//find best locn based on obj

Obj_Locn(ObjX, ObjY, ox, oy);
disto = sqrt(Dist(X, Y, ObjX, ObjY));
pri[3] = o * disto;

//best location based on highest pri of the four:

b = 0;
best = pri[0];
for(i=1; i<4; i++)           //find highest priority move
{
    if(pri[i] > best)
{
    best = pri[i];
    b = i;
}
}

```

```

}
}

switch(b)
{
  case 0: if(distf != 0.0)
    {
Dest_X = Rnd(X + (fx - X)/distf);
Dest_Y = Rnd(Y + (fy - Y)/distf);
    }
    break;

  case 1: if(dista != 0.0)
    {
Dest_X = Rnd(X + (ax - X)/dista);
Dest_Y = Rnd(Y + (ay - Y)/dista);
    }
    break;

  case 2: if(diste != 0.0)
    {
Dest_X = Rnd(X + (ex - X)/diste);
Dest_Y = Rnd(Y + (ey - Y)/diste);
    }
    break;

  case 3: if(disto != 0.0)
    {
Dest_X = Rnd(X + (ox - X)/sqr(disto));
Dest_Y = Rnd(Y + (oy - Y)/sqr(disto));
    }
    break;

  default: cerr << "problem in ch_best_art_move" << endl;
}

if(Dest_X == Last_X && Dest_Y == Last_Y)
    //if moving back to previous spot, don't
    {
    Dest_X = X;
    Dest_Y = Y;
    }

//set moving flag

```

```

    if(Dest_X != X || Dest_Y != Y) Moving = true;
    else Moving = false;

    //determine next update time
    dist = Dist(X, Y, Dest_X, Dest_Y);

    //if sitting on best spot, stay 1/shots per min
    if(dist == 0.0) NextTime.Mv = NextTime.Mv + Reload;

    //else compute next event time
    else NextTime.Mv = NextTime.Mv + (dist/Move_Rate);
    return true;
}

bool Art::ClearPtrs()
{
    TGT* temp;
    temp = Fr;
    while (temp != NULL)                //there were fr's in area
    {
        Fr = Fr->Next;
        temp->Next = NULL;
        delete temp;
        temp = Fr;
    }
    temp = En;
    while (temp != NULL)                //there were en's in area
    {
        En = En->Next;
        temp->Next = NULL;
        delete temp;
        temp = En;
    }

    temp = Arty;
    while (temp != NULL)                //there were arty in area
    {
        Arty = Arty->Next;
        temp->Next = NULL;
        delete temp;
        temp = Arty;
    }

    enctr = 0;
    frctr = 1;
}

```

```

    arctr = 0;
    return true;
}

double Art::Fr_Locn(long frX, long frY, long &fx, long &fy)
{
    double dist = Dist( X, Y, frX, frY);
    if(dist > G_Rg/3)
    {
        fx = frX;                //obj attracts arty
        fy = frY;
    }
    else                          //best to stay in location
    {
        fx = X;
        fy = Y;
    }
    fx = max (0, fx);            //stay on game board
    fy = max (0, fy);
    fx = min (fx, Landscape_Size-1);
    fy = min (fy, Landscape_Size-1);
    return 1.0;
}

double Art::Ar_Locn(long arX, long arY, long &ax, long &ay)
{
    double p;
    p = Uniform(0,1);

    //best locn is offset from the friendly art by the opt dist
    if( (arX > X) || ((arX == X) && (p < 0.5))) ax = Rnd( double(arX) -
        double(O_Dist) );                //fr is below
    else ax = Rnd( double(arX) + double(O_Dist) );                //else above

    p = Uniform(0,1);
    if( (arY > Y) || ((arX == X) && (p < 0.5))) ay = Rnd( double(arY) -
        double(O_Dist) );                //fr is right
    else if(arY < Y) ay = Rnd( double(arY) + double(O_Dist) ); //or left
    else ay = arY;                    //else on-line

    ax = max(0, ax);                //stay on game board
    ay = max(0, ay);
    ax = min(ax, Landscape_Size-1);
    ay = min(ay, Landscape_Size-1);
    return 1.0;
}

```

```

}

double Art::En_Locn(long enX, long enY, long &ex, long &ey)
{
    if(X - enX > 0) ex = X+1;
    else ex = X-1;

    if(Y - enY > 0) ey = Y+1;
    else ey = Y-1;

    return 1.0;
}

double Art::Obj_Locn(long obX, long obY, long &ox, long &oy)
{
    double dist = Dist( X, Y, obX, obY);
    if(dist > 0.667*G_Rg)
    {
        ox = obX;           //obj attracts arty
        oy = obY;
    }
    else                   //best to stay in location
    {
        ox = obX;
        oy = obY;
    }
    ox = max (0, ox);     //stay on game board
    oy = max (0, oy);
    ox = min (ox, Landscape_Size-1);
    oy = min (oy, Landscape_Size-1);
    return 1.0;
}

```

```

////////////////////////////////////
//                                                                    //
// art.h:          Header file for the tank class, derived from //
//                the veh class                                //
////////////////////////////////////

#ifndef ART_H
#define ART_H
#include "veh.h"
#include "rvgs.h"
#include "rngs.h"
#include "Param.h"
#include "land.h"
#include <math.h>
#include <iostream>

#define STRICT

class Art : public Veh
{
private:

public:

    Art(long x, long y, char clr, char arr[], int n);
    ~Art();
    //locate best move based on:
    //fr's, en, obj
    bool Choose_Next_Move(long f, long a, long e, long o);
    double Fr_Locn(long frX, long frY, long &fx, long &fy);

    double Ar_Locn(long frX, long frY, long &fx, long &fy);
    double En_Locn(long enX, long enY, long &ex, long &ey);
    double Obj_Locn(long obX, long obY, long &ox, long &oy);
    bool ClearPtrs();

};

#endif

```

```

////////////////////////////////////
//                                                                    //
//      land.cpp:  Contains the Landscape data for Cbt Sim.  //
//                                                                    //
////////////////////////////////////

#include "land.h"
#include "Param.h"
#include "RParam.h"
#include <cmath>
#include <stdio.h>
#include <fstream>
#define sqr(x)  ((x)*(x))

World Cell[Landscape_Size][Landscape_Size]; //Instantiates the landscape

bool Init_World()
{
    long i = 0;
    long j = 0;

    for (i=0; i < Landscape_Size; i++)
    {
        for (j=0; j < Landscape_Size; j++)
        {
            Cell[i][j].Number_On_Point = 0;
            Cell[i][j].Color = 'u';
            Cell[i][j].Type = 5;
            Cell[i][j].Occ = NULL;
        }
    }

    return true;
}

void clearCell(long x, long y)
{
    Cell[x][y].Number_On_Point = 0;
    Cell[x][y].Color = 'u';
    Cell[x][y].Type = 5;
    Cell[x][y].Occ = NULL;
}

```

```

////////////////////////////////////
//                                                                    //
//   Common functions needed by all                                    //
//                                                                    //
////////////////////////////////////

long max(long i, long j)
{
    if(i > j) return i;
    else return j;
}

long min (long i, long j)
{
    if(i < j) return i;
    else return j;
}

long Rnd(double x)
{
    if(x >= 0)
    {
        if(x > long(x)+0.50) return (long(x)+1);
        else return long(x);
    }
    else
    {
        if(x < long(x)-0.50) return (long(x)-1);
        else return long(x);
    }
}

double Dist(long i, long j, long a, long b) //measures euclidean distance
{
    double dist;
    dist = sqrt(sqr(a-i)+sqr(b-j));
    return dist;
}

long mabs(long x) //returns absolute value
{
    if(x < 0) return -x;
}

```



```
else return x;  
}
```

```

////////////////////////////////////
//                               //
//   land.h: Organizes the Landscape data for Cbt Sim.   //
//                               //
////////////////////////////////////

#include "veh.h"
#include "rvgs.h"
#include "rngs.h"

#ifndef LAND_H
#define LAND_H

struct World
{
    long  Number_On_Point;           //number of occupants on a point
    char  Color;                    //color of occupant, if any
    long  Type;                     //type of occupant, if any
    Veh*  Occ;                      //pointer to occupant
};

bool Init_World();

void clearCell(long x, long y);

long max(long i, long j);

long min (long i, long j);

long Rnd(double x);

double Dist(long i, long j, long a, long b);

long mabs(long x);

#endif

```

```

/* -----
 * This is an ANSI C library for multi-stream random number generation.
 * The use of this library is recommended as a replacement for the ANSI C
 * rand() and srand() functions, particularly in simulation applications
 * where the statistical 'goodness' of the random number generator is
 * important. The library supplies 256 streams of random numbers; use
 * SelectStream(s) to switch between streams indexed s = 0,1,...,255.
 *
 * The streams must be initialized. The recommended way to do this is by
 * using the function PlantSeeds(x) with the value of x used to initialize
 * the default stream and all other streams initialized automatically with
 * values dependent on the value of x. The following convention is used
 * to initialize the default stream:
 *   if x > 0 then x is the state
 *   if x < 0 then the state is obtained from the system clock
 *   if x = 0 then the state is to be supplied interactively.
 *
 * The generator used in this library is a so-called 'Lehmer random number
 * generator' which returns a pseudo-random number uniformly distributed
 * 0.0 and 1.0. The period is (m - 1) where m = 2,147,483,647 and the
 * smallest and largest possible values are (1 / m) and 1 - (1 / m)
 * respectively. For more details see:
 *
 *       "Random Number Generators: Good Ones Are Hard To Find"
 *           Steve Park and Keith Miller
 *       Communications of the ACM, October 1988
 *
 * Name           : rngs.c (Random Number Generation - Multiple Streams)
 * Authors        : Steve Park & Dave Geyer
 * Language       : ANSI C
 * Latest Revision : 09-22-98
 * -----
 */

#include <stdio.h>
#include <time.h>
#include "rngs.h"

#define MODULUS      2147483647 /* DON'T CHANGE THIS VALUE */
#define MULTIPLIER  48271      /* DON'T CHANGE THIS VALUE */
#define CHECK        399268537 /* DON'T CHANGE THIS VALUE */
#define STREAMS     256        /* # of streams, DON'T CHANGE THIS VALUE */
#define A256        22925      /* jump multiplier, DON'T CHANGE THIS VAL*/
#define DEFAULT     123456789 /* initial seed, use 0< DEFAULT < MODULUS*/

```

```

static long seed[STREAMS] = {DEFAULT}; /* current state of each stream*/
static int  stream        = 0;         /* stream index, 0 is the default */
static int  initialized   = 0;         /* test for stream initialization */

    double Random(void)
/* -----
 * Random returns a pseudo-random real number uniformly distributed
 * between 0.0 and 1.0.
 * -----
 */
{
    const long Q = MODULUS / MULTIPLIER;
    const long R = MODULUS % MULTIPLIER;
        long t;

    t = MULTIPLIER * (seed[stream] % Q) - R * (seed[stream] / Q);
    if (t > 0)
        seed[stream] = t;
    else
        seed[stream] = t + MODULUS;
    return ((double) seed[stream] / MODULUS);
}

    void PlantSeeds(long x)
/* -----
 * Use this function to set the state of all the random number generator
 * streams by "planting" a sequence of states (seeds), one per stream,
 * with all states dictated by the state of the default stream.
 * The sequence of planted states is separated one from the next by
 * 8,367,782 calls to Random().
 * -----
 */
{
    const long Q = MODULUS / A256;
    const long R = MODULUS % A256;
        int  j;
        int  s;

    initialized = 1;
    s = stream; /* remember the current stream */
    SelectStream(0); /* change to stream 0 */
    PutSeed(x); /* set seed[0] */
    stream = s; /* reset the current stream */
}

```

```

for (j = 1; j < STREAMS; j++) {
    x = A256 * (seed[j - 1] % Q) - R * (seed[j - 1] / Q);
    if (x > 0)
        seed[j] = x;
    else
        seed[j] = x + MODULUS;
}
}

void PutSeed(long x)
/* -----
 * Use this function to set the state of the current random number
 * generator stream according to the following conventions:
 *   if x > 0 then x is the state (unless too large)
 *   if x < 0 then the state is obtained from the system clock
 *   if x = 0 then the state is to be supplied interactively
 * -----
 */
{
    char ok = 0;

    if (x > 0)
        x = x % MODULUS;
    if (x < 0)
        x = ((unsigned long) time((time_t *) NULL)) % MODULUS;
    if (x == 0)
        while (!ok) {
            printf("\nEnter a positive integer seed (9 digits or less) >> ");
            scanf("%ld", &x);
            ok = (0 < x) && (x < MODULUS);
            if (!ok)
                printf("\nInput out of range ... try again\n");
        }
    seed[stream] = x;
}

void GetSeed(long *x)
/* -----
 * Use this function to get the state of the current random number
 * generator stream.
 * -----
 */
{

```

```

    *x = seed[stream];
}

    void SelectStream(int index)
/* -----
 * Use this function to set the current random number generator
 * stream -- that stream from which the next random number will come.
 * -----
 */
{
    stream = ((unsigned int) index) % STREAMS;
    if ((initialized == 0) && (stream != 0)) /* protect against      */
        PlantSeeds(DEFAULT);                /* un-initialized streams */
}

    void TestRandom(void)
/* -----
 * Use this (optional) function to test for a correct implementation.
 * -----
 */
{
    long    i;
    long    x;
    double  u;
    char    ok = 0;

    SelectStream(0);                /* select the default stream */
    PutSeed(1);                     /* and set the state to 1    */
    for(i = 0; i < 10000; i++)
        u = Random();
    GetSeed(&x);                    /* get the new state value  */
    ok = (x == CHECK);              /* and check for correctness */

    SelectStream(1);                /* select stream 1          */
    PlantSeeds(1);                  /* set the state of all streams */
    GetSeed(&x);                    /* get the state of stream 1 */
    ok = ok && (x == A256);          /* x should be the jump multiplier */
    if (ok)
        printf("\n The implementation of rngs.c is correct.\n\n");
    else
        printf("\n\a ERROR-the implementation of rngs.c is not correct.\n\n");
}

```

```
/* -----  
 * Name           : rngs.h (header file for the library file rngs.c)  
 * Author        : Steve Park & Dave Geyer  
 * Language      : ANSI C  
 * Latest Revision : 09-22-98  
 * -----  
 */  
  
#if !defined( _RNGS_ )  
#define _RNGS_  
  
double Random(void);  
void PlantSeeds(long x);  
void GetSeed(long *x);  
void PutSeed(long x);  
void SelectStream(int index);  
void TestRandom(void);  
  
#endif
```

```

/* -----
 * This is an ANSI C library for generating random variates from six
 * discrete distributions
 *
 * Generator          Range (x)      Mean          Variance
 *
 * Bernoulli(p)      x = 0,1        p              p*(1-p)
 * Binomial(n, p)    x = 0,...,n      n*p            n*p*(1-p)
 * Equilikely(a, b) x = a,...,b      (a+b)/2        ((b-a+1)*(b-a+1)-1)/12
 * Geometric(p)      x = 0,...        p/(1-p)        p/((1-p)*(1-p))
 * Pascal(n, p)      x = 0,...        n*p/(1-p)      n*p/((1-p)*(1-p))
 * Poisson(m)        x = 0,...        m              m
 *
 * and seven continuous distributions
 *
 * Uniform(a, b)     a < x < b        (a + b)/2      (b - a)*(b - a)/12
 * Exponential(m)    x > 0            m              m*m
 * Erlang(n, b)      x > 0            n*b            n*b*b
 * Normal(m, s)      all x            m              s*s
 * Lognormal(a, b)   x > 0            see below
 * Chisquare(n)      x > 0            n              2*n
 * Student(n)        all x            0 (n > 1)     n/(n - 2) (n > 2)
 *
 * For the a Lognormal(a, b) random variable, the mean and variance are
 *
 * mean = exp(a + 0.5*b*b)
 * variance = (exp(b*b) - 1) * exp(2*a + b*b)
 *
 * Name           : rvgs.c (Random Variate GeneratorS)
 * Author          : Steve Park & Dave Geyer
 * Language        : ANSI C
 * Latest Revision : 10-28-98
 * -----
 */

#include <math.h>
#include "rngs.h"
#include "rvgs.h"

long Bernoulli(double p)
/* =====
 * Returns 1 with probability p or 0 with probability 1 - p.
 * NOTE: use 0.0 < p < 1.0
 * =====

```



```

*/
{
  return ((Random() < (1.0 - p)) ? 0 : 1);
}

long Binomial(long n, double p)
/* =====
* Returns a binomial distributed integer between 0 and n inclusive.
* NOTE: use n > 0 and 0.0 < p < 1.0
* =====
*/
{
  long i, x = 0;

  for (i = 0; i < n; i++)
    x += Bernoulli(p);
  return (x);
}

long Equilikely(long a, long b)
/* =====
* Returns an equilikely distributed integer between a and b inclusive.
* NOTE: use a < b
* =====
*/
{
  return (a + (long) ((b - a + 1) * Random()));
}

long Geometric(double p)
/* =====
* Returns a geometric distributed non-negative integer.
* NOTE: use 0.0 < p < 1.0
* =====
*/
{
  return ((long) (log(1.0 - Random()) / log(p)));
}

long Pascal(long n, double p)
/* =====
* Returns a Pascal distributed non-negative integer.
* NOTE: use n > 0 and 0.0 < p < 1.0
* =====
*/

```

```

{
    long i, x = 0;

    for (i = 0; i < n; i++)
        x += Geometric(p);
    return (x);
}

long Poisson(double m)
/* =====
 * Returns a Poisson distributed non-negative integer.
 * NOTE: use m > 0
 * =====
 */
{
    double t = 0.0;
    long x = 0;

    while (t < m) {
        t += Exponential(1.0);
        x++;
    }
    return (x - 1);
}

double Uniform(double a, double b)
/* =====
 * Returns a uniformly distributed real number between a and b.
 * NOTE: use a < b
 * =====
 */
{
    return (a + (b - a) * Random());
}

double Exponential(double m)
/* =====
 * Returns an exponentially distributed positive real number.
 * NOTE: use m > 0.0
 * =====
 */
{
    return (-m * log(1.0 - Random()));
}

```

```

    double Erlang(long n, double b)
/* =====
 * Returns an Erlang distributed positive real number.
 * NOTE: use n > 0 and b > 0.0
 * =====
 */
{
    long i;
    double x = 0.0;

    for (i = 0; i < n; i++)
        x += Exponential(b);
    return (x);
}

    double Normal(double m, double s)
/* =====
 * Returns a normal (Gaussian) distributed real number.
 * NOTE: use s > 0.0
 *
 * Uses a very accurate approximation of the normal idf due to Odeh &
 * Evans, J. Applied Statistics, 1974, vol 23, pp 96-97.
 * =====
 */
{
    const double p0 = 0.322232431088;    const double q0 = 0.099348462606;
    const double p1 = 1.0;                const double q1 = 0.588581570495;
    const double p2 = 0.342242088547;    const double q2 = 0.531103462366;
    const double p3 = 0.204231210245e-1; const double q3 = 0.103537752850;
    const double p4 = 0.453642210148e-4; const double q4 = 0.385607006340e-2;
    double u, t, p, q, z;

    u = Random();
    if (u < 0.5)
        t = sqrt(-2.0 * log(u));
    else
        t = sqrt(-2.0 * log(1.0 - u));
    p = p0 + t * (p1 + t * (p2 + t * (p3 + t * p4)));
    q = q0 + t * (q1 + t * (q2 + t * (q3 + t * q4)));
    if (u < 0.5)
        z = (p / q) - t;
    else
        z = t - (p / q);
    return (m + s * z);
}

```

```

    double Lognormal(double a, double b)
/* =====
 * Returns a lognormal distributed positive real number.
 * NOTE: use b > 0.0
 * =====
 */
{
    return (exp(a + b * Normal(0.0, 1.0)));
}

    double Chisquare(long n)
/* =====
 * Returns a chi-square distributed positive real number.
 * NOTE: use n > 0
 * =====
 */
{
    long    i;
    double z, x = 0.0;

    for (i = 0; i < n; i++) {
        z = Normal(0.0, 1.0);
        x += z * z;
    }
    return (x);
}

    double Student(long n)
/* =====
 * Returns a student-t distributed real number.
 * NOTE: use n > 0
 * =====
 */
{
    return (Normal(0.0, 1.0) / sqrt(Chisquare(n) / n));
}

```

```
/* -----  
 * Name          : rvgs.h (header file for the library rvgs.c)  
 * Author        : Steve Park & Dave Geyer  
 * Language      : ANSI C  
 * Latest Revision : 11-03-96  
 * -----  
 */  
  
#ifndef RVGS_H  
#define RVGS_H  
  
long Bernoulli(double p);  
long Binomial(long n, double p);  
long Equilikely(long a, long b);  
long Geometric(double p);  
long Pascal(long n, double p);  
long Poisson(double m);  
  
double Uniform(double a, double b);  
double Exponential(double m);  
double Erlang(long n, double b);  
double Normal(double m, double s);  
double Lognormal(double a, double b);  
double Chisquare(long n);  
double Student(long n);  
  
#endif
```

```

/* -----
* This is an ANSI C library that can be used to evaluate the probability
* density functions (pdf's), cumulative distribution functions (cdf's),
* and inverse distribution functions (idf's) for a variety of discrete
* and continuous random variables.
*
* The following notational conventions are used
*     x : possible value of the random variable
*     u : real variable (probability) between 0.0 and 1.0
*     a, b, n, p, m, s : distribution-specific parameters
*
* There are pdf's, cdf's and idf's for 6 discrete random variables
*
* Random Variable      Range (x)  Mean      Variance
*
* Bernoulli(p)         0..1      p          p*(1-p)
* Binomial(n, p)       0..n      n*p        n*p*(1-p)
* Equilikely(a, b)    a..b      (a+b)/2    ((b-a+1)*(b-a+1)-1)/12
* Geometric(p)         0...      p/(1-p)    p/((1-p)*(1-p))
* Pascal(n, p)         0...      n*p/(1-p)  n*p/((1-p)*(1-p))
* Poisson(m)           0...      m          m
*
* and for 7 continuous random variables
*
* Uniform(a, b)        a < x < b  (a+b)/2    (b-a)*(b-a)/12
* Exponential(m)       x > 0     m          m*m
* Erlang(n, b)         x > 0     n*b        n*b*b
* Normal(m, s)         all x     m          s*s
* Lognormal(a, b)      x > 0     see below
* Chisquare(n)         x > 0     n          2*n
* Student(n)           all x     0 (n > 1)  n/(n-2)   (n > 2)
*
* For the Lognormal(a, b), the mean and variance are
*
*          mean = Exp(a + 0.5*b*b)
*          variance = (Exp(b*b) - 1)*Exp(2*a + b*b)
*
* Name           : rvms.c (Random Variable Models)
* Author          : Steve Park & Dave Geyer
* Language        : ANSI C
* Latest Revision : 11-22-97
* -----
*/

#include <math.h>

```

```

#include "rvms.h"

#define TINY    1.0e-10
#define SQRT2PI 2.506628274631          /* sqrt(2 * pi) */

static double pdfStandard(double x);
static double cdfStandard(double x);
static double idfStandard(double u);
static double LogGamma(double a);
static double LogBeta(double a, double b);
static double InGamma(double a, double b);
static double InBeta(double a, double b, double x);

    double pdfBernoulli(double p, long x)
/* =====
* NOTE: use 0.0 < p < 1.0 and 0 <= x <= 1
* =====
*/
{
    return ((x == 0) ? 1.0 - p : p);
}

    double cdfBernoulli(double p, long x)
/* =====
* NOTE: use 0.0 < p < 1.0 and 0 <= x <= 1
* =====
*/
{
    return ((x == 0) ? 1.0 - p : 1.0);
}

    long idfBernoulli(double p, double u)
/* =====
* NOTE: use 0.0 < p < 1.0 and 0.0 < u < 1.0
* =====
*/
{
    return ((u < 1.0 - p) ? 0 : 1);
}

    double pdfEquillikely(long a, long b, long x)
/* =====
* NOTE: use a <= x <= b
* =====

```

```

*/
{
    return (1.0 / (b - a + 1.0));
}

double cdfEquilikely(long a, long b, long x)
/* =====
* NOTE: use a <= x <= b
* =====
*/
{
    return ((x - a + 1.0) / (b - a + 1.0));
}

long idfEquilikely(long a, long b, double u)
/* =====
* NOTE: use a <= b and 0.0 < u < 1.0
* =====
*/
{
    return (a + (long) (u * (b - a + 1)));
}

double pdfBinomial(long n, double p, long x)
/* =====
* NOTE: use 0 <= x <= n and 0.0 < p < 1.0
* =====
*/
{
    double s, t;

    s = LogChoose(n, x);
    t = x * log(p) + (n - x) * log(1.0 - p);
    return (exp(s + t));
}

double cdfBinomial(long n, double p, long x)
/* =====
* NOTE: use 0 <= x <= n and 0.0 < p < 1.0
* =====
*/
{
    if (x < n)
        return (1.0 - InBeta(x + 1, n - x, p));
    else

```



```

    return (1.0);
}

long idfBinomial(long n, double p, double u)
/* =====
 * NOTE: use 0 <= n, 0.0 < p < 1.0 and 0.0 < u < 1.0
 * =====
 */
{
    long x = (long) (n * p);    /* start searching at the mean */

    if (cdfBinomial(n, p, x) <= u)
        while (cdfBinomial(n, p, x) <= u)
            x++;
    else if (cdfBinomial(n, p, 0) <= u)
        while (cdfBinomial(n, p, x - 1) > u)
            x--;
    else
        x = 0;
    return (x);
}

double pdfGeometric(double p, long x)
/* =====
 * NOTE: use 0.0 < p < 1.0 and x >= 0
 * =====
 */
{
    return ((1.0 - p) * exp(x * log(p)));
}

double cdfGeometric(double p, long x)
/* =====
 * NOTE: use 0.0 < p < 1.0 and x >= 0
 * =====
 */
{
    return (1.0 - exp((x + 1) * log(p)));
}

long idfGeometric(double p, double u)
/* =====
 * NOTE: use 0.0 < p < 1.0 and 0.0 < u < 1.0
 * =====
 */

```

```

{
    return ((long) (log(1.0 - u) / log(p)));
}

double pdfPascal(long n, double p, long x)
/* =====
 * NOTE: use n >= 1, 0.0 < p < 1.0, and x >= 0
 * =====
 */
{
    double s, t;

    s = LogChoose(n + x - 1, x);
    t = x * log(p) + n * log(1.0 - p);
    return (exp(s + t));
}

double cdfPascal(long n, double p, long x)
/* =====
 * NOTE: use n >= 1, 0.0 < p < 1.0, and x >= 0
 * =====
 */
{
    return (1.0 - InBeta(x + 1, n, p));
}

long idfPascal(long n, double p, double u)
/* =====
 * NOTE: use n >= 1, 0.0 < p < 1.0, and 0.0 < u < 1.0
 * =====
 */
{
    long x = (long) (n * p / (1.0 - p));
                                /* start searching at the mean */

    if (cdfPascal(n, p, x) <= u)
        while (cdfPascal(n, p, x) <= u)
            x++;
    else if (cdfPascal(n, p, 0) <= u)
        while (cdfPascal(n, p, x - 1) > u)
            x--;
    else
        x = 0;
    return (x);
}

```

```

    double pdfPoisson(double m, long x)
/* =====
* NOTE: use m > 0 and x >= 0
* =====
*/
{
    double t;

    t = - m + x * log(m) - LogFactorial(x);
    return (exp(t));
}

    double cdfPoisson(double m, long x)
/* =====
* NOTE: use m > 0 and x >= 0
* =====
*/
{
    return (1.0 - InGamma(x + 1, m));
}

    long idfPoisson(double m, double u)
/* =====
* NOTE: use m > 0 and 0.0 < u < 1.0
* =====
*/
{
    long x = (long) m;                                /* start searching at the mean */

    if (cdfPoisson(m, x) <= u)
        while (cdfPoisson(m, x) <= u)
            x++;
    else if (cdfPoisson(m, 0) <= u)
        while (cdfPoisson(m, x - 1) > u)
            x--;
    else
        x = 0;
    return (x);
}

    double pdfUniform(double a, double b, double x)
/* =====
* NOTE: use a < x < b
* =====

```

```

*/
{
    return (1.0 / (b - a));
}

double cdfUniform(double a, double b, double x)
/* =====
* NOTE: use a < x < b
* =====
*/
{
    return ((x - a) / (b - a));
}

double idfUniform(double a, double b, double u)
/* =====
* NOTE: use a < b and 0.0 < u < 1.0
* =====
*/
{
    return (a + (b - a) * u);
}

double pdfExponential(double m, double x)
/* =====
* NOTE: use m > 0 and x > 0
* =====
*/
{
    return ((1.0 / m) * exp(- x / m));
}

double cdfExponential(double m, double x)
/* =====
* NOTE: use m > 0 and x > 0
* =====
*/
{
    return (1.0 - exp(- x / m));
}

double idfExponential(double m, double u)
/* =====
* NOTE: use m > 0 and 0.0 < u < 1.0
* =====

```

```

*/
{
    return (- m * log(1.0 - u));
}

double pdfErlang(long n, double b, double x)
/* =====
* NOTE: use n >= 1, b > 0, and x > 0
* =====
*/
{
    double t;

    t = (n - 1) * log(x / b) - (x / b) - log(b) - LogGamma(n);
    return (exp(t));
}

double cdfErlang(long n, double b, double x)
/* =====
* NOTE: use n >= 1, b > 0, and x > 0
* =====
*/
{
    return (InGamma(n, x / b));
}

double idfErlang(long n, double b, double u)
/* =====
* NOTE: use n >= 1, b > 0 and 0.0 < u < 1.0
* =====
*/
{
    double t, x = n * b;          /* initialize to the mean, then */

    do {                          /* use Newton-Raphson iteration */
        t = x;
        x = t + (u - cdfErlang(n, b, t)) / pdfErlang(n, b, t);
        if (x <= 0.0)
            x = 0.5 * t;
    } while (fabs(x - t) >= TINY);
    return (x);
}

static double pdfStandard(double x)
/* =====

```

```

* NOTE: x can be any value
* =====
*/
{
    return (exp(- 0.5 * x * x) / SQRT2PI);
}

static double cdfStandard(double x)
/* =====
* NOTE: x can be any value
* =====
*/
{
    double t;

    t = InGamma(0.5, 0.5 * x * x);
    if (x < 0.0)
        return (0.5 * (1.0 - t));
    else
        return (0.5 * (1.0 + t));
}

static double idfStandard(double u)
/* =====
* NOTE: 0.0 < u < 1.0
* =====
*/
{
    double t, x = 0.0;          /* initialize to the mean, then */

    do {                        /* use Newton-Raphson iteration */
        t = x;
        x = t + (u - cdfStandard(t)) / pdfStandard(t);
    } while (fabs(x - t) >= TINY);
    return (x);
}

double pdfNormal(double m, double s, double x)
/* =====
* NOTE: x and m can be any value, but s > 0.0
* =====
*/
{
    double t = (x - m) / s;

```

```

    return (pdfStandard(t) / s);
}

double cdfNormal(double m, double s, double x)
/* =====
* NOTE: x and m can be any value, but s > 0.0
* =====
*/
{
    double t = (x - m) / s;

    return (cdfStandard(t));
}

double idfNormal(double m, double s, double u)
/* =====
* NOTE: m can be any value, but s > 0.0 and 0.0 < u < 1.0
* =====
*/
{
    return (m + s * idfStandard(u));
}

double pdfLognormal(double a, double b, double x)
/* =====
* NOTE: a can have any value, but b > 0.0 and x > 0.0
* =====
*/
{
    double t = (log(x) - a) / b;

    return (pdfStandard(t) / (b * x));
}

double cdfLognormal(double a, double b, double x)
/* =====
* NOTE: a can have any value, but b > 0.0 and x > 0.0
* =====
*/
{
    double t = (log(x) - a) / b;

    return (cdfStandard(t));
}

```

```

    double idfLognormal(double a, double b, double u)
/* =====
* NOTE: a can have any value, but b > 0.0 and 0.0 < u < 1.0
* =====
*/
{
    double t;

    t = a + b * idfStandard(u);
    return (exp(t));
}

    double pdfChisquare(long n, double x)
/* =====
* NOTE: use n >= 1 and x > 0.0
* =====
*/
{
    double t, s = n / 2.0;

    t = (s - 1.0) * log(x / 2.0) - (x / 2.0) - log(2.0) - LogGamma(s);
    return (exp(t));
}

    double cdfChisquare(long n, double x)
/* =====
* NOTE: use n >= 1 and x > 0.0
* =====
*/
{
    return (InGamma(n / 2.0, x / 2));
}

    double idfChisquare(long n, double u)
/* =====
* NOTE: use n >= 1 and 0.0 < u < 1.0
* =====
*/
{
    double t, x = n;          /* initialize to the mean, then */

    do {                      /* use Newton-Raphson iteration */
        t = x;
        x = t + (u - cdfChisquare(n, t)) / pdfChisquare(n, t);
        if (x <= 0.0)

```



```

        x = 0.5 * t;
    } while (fabs(x - t) >= TINY);
    return (x);
}

double pdfStudent(long n, double x)
/* =====
* NOTE: use n >= 1 and x > 0.0
* =====
*/
{
    double s, t;

    s = -0.5 * (n + 1) * log(1.0 + ((x * x) / (double) n));
    t = -LogBeta(0.5, n / 2.0);
    return (exp(s + t) / sqrt((double) n));
}

double cdfStudent(long n, double x)
/* =====
* NOTE: use n >= 1 and x > 0.0
* =====
*/
{
    double s, t;

    t = (x * x) / (n + x * x);
    s = InBeta(0.5, n / 2.0, t);
    if (x >= 0.0)
        return (0.5 * (1.0 + s));
    else
        return (0.5 * (1.0 - s));
}

double idfStudent(long n, double u)
/* =====
* NOTE: use n >= 1 and 0.0 < u < 1.0
* =====
*/
{
    double t, x = 0.0;          /* initialize to the mean, then */

    do {                        /* use Newton-Raphson iteration */
        t = x;
        x = t + (u - cdfStudent(n, t)) / pdfStudent(n, t);
    }
}

```

```

    } while (fabs(x - t) >= TINY);
    return (x);
}

/* =====
 * The six functions that follow are a 'special function' mini-library
 * used to support the evaluation of pdf, cdf and idf functions.
 * =====
 */

static double LogGamma(double a)
/* =====
 * LogGamma returns the natural log of the gamma function.
 * NOTE: use a > 0.0
 *
 * The algorithm used to evaluate the natural log of the gamma function is
 * based on an approximation by C. Lanczos, SIAM J. Numerical Analysis, B,
 * vol 1, 1964. The constants have been selected to yield a relative error
 * which is less than 2.0e-10 for all positive values of the parameter a.
 * =====
 */
{
    double s[6], sum, temp;
    int    i;

    s[0] = 76.180091729406 / a;
    s[1] = -86.505320327112 / (a + 1.0);
    s[2] = 24.014098222230 / (a + 2.0);
    s[3] = -1.231739516140 / (a + 3.0);
    s[4] = 0.001208580030 / (a + 4.0);
    s[5] = -0.000005363820 / (a + 5.0);
    sum = 1.000000000178;
    for (i = 0; i < 6; i++)
        sum += s[i];
    temp = (a - 0.5) * log(a + 4.5) - (a + 4.5) + log(SQRT2PI * sum);
    return (temp);
}

double LogFactorial(long n)
/* =====
 * LogFactorial(n) returns the natural log of n!
 * NOTE: use n >= 0
 *
 * The algorithm used to evaluate the natural log of n! is based on a
 * simple equation which relates the gamma and factorial functions.
 */

```

```

* =====
*/
{
    return (LogGamma(n + 1));
}

    static double LogBeta(double a, double b)
/* =====
* LogBeta returns the natural log of the beta function.
* NOTE: use a > 0.0 and b > 0.0
*
* The algorithm used to evaluate the natural log of the beta function is
* based on a simple equation which relates the gamma and beta functions.
*
*/
{
    return (LogGamma(a) + LogGamma(b) - LogGamma(a + b));
}

    double LogChoose(long n, long m)
/* =====
* LogChoose returns the natural log of the binomial coefficient C(n,m).
* NOTE: use 0 <= m <= n
*
* The algorithm used to evaluate the natural log of a binomial coefficient
* is based on a simple equation which relates the beta function to a
* binomial coefficient.
* =====
*/
{
    if (m > 0)
        return (-LogBeta(m, n - m + 1) - log(m));
    else
        return (0.0);
}

    static double InGamma(double a, double x)
/* =====
* Evaluates the incomplete gamma function.
* NOTE: use a > 0.0 and x >= 0.0
*
* The algorithm used to evaluate the incomplete gamma function is based on
* Algorithm AS 32, J. Applied Statistics, 1970, by G. P. Bhattacharjee.
* See also equations 6.5.29 and 6.5.31 in the Handbook of Mathematical
* Functions, Abramowitz and Stegun (editors). The absolute error is less

```

```

* than 1e-10 for all non-negative values of x.
* =====
*/
{
double t, sum, term, factor, f, g, c[2], p[3], q[3];
long n;

if (x > 0.0)
    factor = exp(-x + a * log(x) - LogGamma(a));
else
    factor = 0.0;
if (x < a + 1.0) {          /* evaluate as an infinite series - */
    t = a;                  /* A & S equation 6.5.29          */
    term = 1.0 / a;
    sum = term;
    while (term >= TINY * sum) { /* sum until 'term' is small */
        t++;
        term *= x / t;
        sum += term;
    }
    return (factor * sum);
}
else {                      /* evaluate as a continued fraction - */
    p[0] = 0.0;             /* A & S eqn 6.5.31 with the extended */
    q[0] = 1.0;             /* pattern 2-a, 2, 3-a, 3, 4-a, 4,... */
    p[1] = 1.0;            /* - see also A & S sec 3.10, eqn (3) */
    q[1] = x;
    f = p[1] / q[1];
    n = 0;
    do {                    /* recursively generate the continued */
        g = f;              /* fraction 'f' until two consecutive */
        n++;                /* values are small                    */
        if ((n % 2) > 0) {
            c[0] = ((double) (n + 1) / 2) - a;
            c[1] = 1.0;
        }
        else {
            c[0] = (double) n / 2;
            c[1] = x;
        }
        p[2] = c[1] * p[1] + c[0] * p[0];
        q[2] = c[1] * q[1] + c[0] * q[0];
        if (q[2] != 0.0) { /* rescale to avoid overflow */
            p[0] = p[1] / q[2];
            q[0] = q[1] / q[2];
        }
    }
}
}

```

```

        p[1] = p[2] / q[2];
        q[1] = 1.0;
        f    = p[1];
    }
} while ((fabs(f - g) >= TINY) || (q[1] != 1.0));
return (1.0 - factor * f);
}
}

static double InBeta(double a, double b, double x)
/* =====
* Evaluates the incomplete beta function.
* NOTE: use a > 0.0, b > 0.0 and 0.0 <= x <= 1.0
*
* The algorithm used to evaluate the incomplete beta function is based on
* equation 26.5.8 in the Handbook of Mathematical Functions, Abramowitz
* and Stegun (editors). The absolute error is less than 1e-10 for all x
* between 0 and 1.
* =====
*/
{
    double t, factor, f, g, c, p[3], q[3];
    int    swap;
    long   n;

    if (x > (a + 1.0) / (a + b + 1.0)) { /* to accelerate convergence */
        swap = 1;                          /* complement x and swap a & b */
        x    = 1.0 - x;
        t    = a;
        a    = b;
        b    = t;
    }
    else /* do nothing */
        swap = 0;
    if (x > 0)
        factor = exp(a * log(x) + b * log(1.0 - x) - LogBeta(a,b)) / a;
    else
        factor = 0.0;
    p[0] = 0.0;
    q[0] = 1.0;
    p[1] = 1.0;
    q[1] = 1.0;
    f    = p[1] / q[1];
    n    = 0;
    do { /* recursively generate the continued */

```

```

g = f;                                /* fraction 'f' until two consecutive */
n++;                                  /* values are small */
if ((n % 2) > 0) {
    t = (double) (n - 1) / 2;
    c = -(a + t) * (a + b + t) * x / ((a + n - 1.0) * (a + n));
}
else {
    t = (double) n / 2;
    c = t * (b - t) * x / ((a + n - 1.0) * (a + n));
}
p[2] = p[1] + c * p[0];
q[2] = q[1] + c * q[0];
if (q[2] != 0.0) {                    /* rescale to avoid overflow */
    p[0] = p[1] / q[2];
    q[0] = q[1] / q[2];
    p[1] = p[2] / q[2];
    q[1] = 1.0;
    f = p[1];
}
} while ((fabs(f - g) >= TINY) || (q[1] != 1.0));
if (swap)
    return (1.0 - factor * f);
else
    return (factor * f);
}

```

```

/* -----
 * Name           : rvms.h (header file for the library rvms.c)
 * Author        : Steve Park & Dave Geyer
 * Language      : ANSI C
 * Latest Revision : 11-02-96
 * -----
 */

#if !defined( _RVMS_ )
#define _RVMS_

double LogFactorial(long n);
double LogChoose(long n, long m);

double pdfBernoulli(double p, long x);
double cdfBernoulli(double p, long x);
long idfBernoulli(double p, double u);

double pdfEquilikely(long a, long b, long x);
double cdfEquilikely(long a, long b, long x);
long idfEquilikely(long a, long b, double u);

double pdfBinomial(long n, double p, long x);
double cdfBinomial(long n, double p, long x);
long idfBinomial(long n, double p, double u);

double pdfGeometric(double p, long x);
double cdfGeometric(double p, long x);
long idfGeometric(double p, double u);

double pdfPascal(long n, double p, long x);
double cdfPascal(long n, double p, long x);
long idfPascal(long n, double p, double u);

double pdfPoisson(double m, long x);
double cdfPoisson(double m, long x);
long idfPoisson(double m, double u);

double pdfUniform(double a, double b, double x);
double cdfUniform(double a, double b, double x);
double idfUniform(double a, double b, double u);

double pdfExponential(double m, double x);
double cdfExponential(double m, double x);
double idfExponential(double m, double u);

```

```
double pdfErlang(long n, double b, double x);
double cdfErlang(long n, double b, double x);
double idfErlang(long n, double b, double u);

double pdfNormal(double m, double s, double x);
double cdfNormal(double m, double s, double x);
double idfNormal(double m, double s, double u);

double pdfLognormal(double a, double b, double x);
double cdfLognormal(double a, double b, double x);
double idfLognormal(double a, double b, double u);

double pdfChisquare(long n, double x);
double cdfChisquare(long n, double x);
double idfChisquare(long n, double u);

double pdfStudent(long n, double x);
double cdfStudent(long n, double x);
double idfStudent(long n, double u);

#endif
```



```
////////////////////////////////////  
//                                                                    //  
//          Input file for the mule          //  
//                                                                    //  
////////////////////////////////////  
  
0111110000000001010010110111001011111011000011011110000110100  
10100010000111111000011110000011000011000011101010001101010000  
10000100011101110011000011000101000111110000111000110101001110  
11001100011010100000010001110000110000101110011001011100
```

Appendix B

Code for the Co-evolutionary

Genetic Algorithm

The following code controls the co-evolutionary genetic algorithm. This requires a script to run the Mule on the desired number of solutions (this dissertation used 30, but did not investigate the “best” number to run) and call the co-evolutionary genetic algorithm. The script should iterate until the solutions converge or a desired number of generations is reached.

APPENDIX B. CODE FOR THE CO-EVOLUTIONARY GENETIC ALGORITHM 229

```

////////////////////////////////////
//                                                                    //
// gacoev.cpp:  Runs GA for the mule                                //
//                                                                    //
// Inputs are 30 x.dat (solution) files and 30 x.result           //
// files. Program reads the files, selects the solutions         //
// from the 30 based on probability drawn from relative           //
// score, performs the crossovers/mutations and outputs 30      //
// new solutions overwriting the 30 x.dat files.                 //
// Old solutions and results are archived for later analysis    //
//                                                                    //
////////////////////////////////////

#include "rngs.h"
#include "rvgs.h"
#include "rvms.h"
#include <stdio.h>
#include <iostream>
#include <ctime>
#include <string>
#include <cstdlib>
#include <cmath>
#include <fstream>

#define pcross 0.6
#define mutrate .001
#define LOC 0.95
#define stop 100000

using namespace std;

string dat[30];
double res[30];
string after[30];
double sel[30];

int main(void);
void inputdat(void);
void inputresult(void);
void chkstop();
void appenddat(void);
void appendresult(void);
void buildsel(void);

```

```

void ga(int);
string mutate(string);
long perm(long, long);
double power(double, long);
void outputdat();

int main()
{
    int a = 0;
    inputdat();
    inputresult();
    chkstop();
    appenddat();
    appendresult();
    buildsel();
    for(a=0; a<29; a=a+2)
        {
            ga(a);
        }
    outputdat();

    return 0;
}

void outputdat()
{
    int n = 0;
    ofstream fout;
    fout.open("0.dat");
    fout << after[n];
    fout.clear();
    fout.close();
    n++;

    fout.open("1.dat");
    fout << after[n];
    fout.clear();
    fout.close();
    n++;

    fout.open("2.dat");
    fout << after[n];
    fout.clear();
    fout.close();
    n++;
}

```

```
fout.open("3.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("4.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("5.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("6.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("7.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("8.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("9.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("10.dat");  
fout << after[n];
```

```
fout.clear();
fout.close();
n++;

fout.open("11.dat");
fout << after[n];
fout.clear();
fout.close();
n++;

fout.open("12.dat");
fout << after[n];
fout.clear();
fout.close();
n++;

fout.open("13.dat");
fout << after[n];
fout.clear();
fout.close();
n++;

fout.open("14.dat");
fout << after[n];
fout.clear();
fout.close();
n++;

fout.open("15.dat");
fout << after[n];
fout.clear();
fout.close();
n++;

fout.open("16.dat");
fout << after[n];
fout.clear();
fout.close();
n++;

fout.open("17.dat");
fout << after[n];
fout.clear();
fout.close();
n++;
```

```
fout.open("18.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("19.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("20.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("21.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("22.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("23.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("24.dat");  
fout << after[n];  
fout.clear();  
fout.close();  
n++;
```

```
fout.open("25.dat");  
fout << after[n];
```

```

    fout.clear();
    fout.close();
    n++;

    fout.open("26.dat");
    fout << after[n];
    fout.clear();
    fout.close();
    n++;

    fout.open("27.dat");
    fout << after[n];
    fout.clear();
    fout.close();
    n++;

    fout.open("28.dat");
    fout << after[n];
    fout.clear();
    fout.close();
    n++;

    fout.open("29.dat");
    fout << after[n];
    fout.clear();
    fout.close();
    n++;
}

void ga(int a)
{
    string s0;
    string s1;
    int x = 0;
    int xover, xover1;
    int ctr;
    char z;
    double p;
    double b = sel[x];

    p = Uniform(0,1);           //select first candidate
    if(p <= b) s0 = dat[0];
    else
        {

```



```

        while(p > b)
    {
        x++;
        b = sel[x];
    }
        s0 = dat[x];
    }

    x = 0;
    b = sel[x];                //select second candidate
    p = Uniform(0,1);
    if(p <= b) s1 = dat[0];
    else
    {
        while(p > b)
    {
        x++;
        b = sel[x];
    }
        s1 = dat[x];
    }

    s0 = mutate(s0);
    s1 = mutate(s1);

    p = Uniform(0,1);          //determine if crossover or not
    if(p > pcross)             //no crossover
    {
        after[a] = s0;
        after[a+1] = s1;
    }

    else
    {
        xover = Equilikely(0,117); //crossover point
        xover1 = Equilikely(118,241);
        for(ctr=xover; ctr<=xover1; ctr++)
    {
        z = s0[ctr];
        s0[ctr] = s1[ctr];
        s1[ctr] = z;
    }
        after[a] = s0;
        after[a+1] = s1;
    }

```

```

}

string mutate(string s)
{
    string str;
    long ctr, m;
    double p, x;

    ctr = 0;
    str = s;
    p = Uniform(0,1);
    x = power(1.0-mutrate, 241);
    if(p <= x) return str;
    else while(x < p)
        {
            ctr++;
            x += perm(241, ctr) * power(mutrate, ctr) *
                power(1-mutrate, 241-ctr);
        }

    while( ctr > 0)
        {
            m = Equilikely(0, 241);
            if(str[m] == '0') str[m] = '1';
            else str[m] = '0';
            ctr--;
        }
    return str;
}

long perm(long x, long n)
{
    long b = 1;
    long b1 = 1;
    while (n > 0)
        {
            b *= x;
            b1 *= n;
            x--;
            n--;
        }
    return b/b1;
}

double power(double x, long n)

```

APPENDIX B. CODE FOR THE CO-EVOLUTIONARY GENETIC ALGORITHM 237

```
{
    double prod = 1.0;
    long i;
    if(n == 0) return 1;
    else for(i=1; i<=n; i++) prod *= x;
    return prod;
}

void buildsel()
{
    int n = 0;
    double y = 0.0;
    double x = 0.0;
    for(n=0; n<30; n++) x += res[n];
    for(n=0; n<30; n++)
    {
        y += res[n];
        sel[n] = y/x;
    }
}

void appenddat()
{
    int n = 0;
    ofstream fout("dat.cdat", ios::out | ios::app);

    for(n=0; n<30; n++) fout << dat[n] << "\n";
    fout << "\n";
    fout.clear();
    fout.close();
}

void appendresult()
{
    int n = 0;
    ofstream fout("result.cdat", ios::out | ios::app);

    for(n=0; n<30; n++) fout << res[n] << "\n";
    fout << "\n";
    fout.clear();
    fout.close();
}

void inputresult()
{
```

```
int n = 0;
double avg = 0.0;

ifstream fin;
fin.open("0.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("1.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("2.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("3.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("4.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("5.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;
```

```
fin.open("6.result");  
fin >> res[n];  
avg += res[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("7.result");  
fin >> res[n];  
avg += res[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("8.result");  
fin >> res[n];  
avg += res[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("9.result");  
fin >> res[n];  
avg += res[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("10.result");  
fin >> res[n];  
avg += res[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("11.result");  
fin >> res[n];  
avg += res[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("12.result");  
fin >> res[n];
```

```
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("13.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("14.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("15.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("16.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("17.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("18.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
```

APPENDIX B. CODE FOR THE CO-EVOLUTIONARY GENETIC ALGORITHM 241

```
n++;

fin.open("19.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("20.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("21.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("22.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("23.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("24.result");
fin >> res[n];
avg += res[n];
fin.clear();
fin.close();
n++;

fin.open("25.result");
```

```
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("26.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("27.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("28.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("29.result");
    fin >> res[n];
    avg += res[n];
    fin.clear();
    fin.close();
}

void chkstop()
{
    int n;
    ifstream fin;
    ofstream fout;
    fin.open("iterator");
    fin >> n;
    fin.clear();
    fin.close();
    n++;
}
```



```
if(n >= stop)
{
    fout.open("quit");
    fout << n;
    fout.clear();
    fout.close();
}
fout.open("iterator");
fout << n;
fout.clear();
fout.close();
}
```

```
void inputdat()
{
    int n = 0;
    ifstream fin;
    fin.open("0.dat");
    fin >> dat[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("1.dat");
    fin >> dat[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("2.dat");
    fin >> dat[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("3.dat");
    fin >> dat[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("4.dat");
    fin >> dat[n];
    fin.clear();
    fin.close();
}
```

```
n++;

fin.open("5.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("6.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("7.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("8.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("9.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("10.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("11.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("12.dat");
```

```
fin >> dat[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("13.dat");  
fin >> dat[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("14.dat");  
fin >> dat[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("15.dat");  
fin >> dat[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("16.dat");  
fin >> dat[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("17.dat");  
fin >> dat[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("18.dat");  
fin >> dat[n];  
fin.clear();  
fin.close();  
n++;
```

```
fin.open("19.dat");  
fin >> dat[n];  
fin.clear();  
fin.close();
```

APPENDIX B. CODE FOR THE CO-EVOLUTIONARY GENETIC ALGORITHM 246

```
n++;

fin.open("20.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("21.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("22.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("23.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("24.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("25.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("26.dat");
fin >> dat[n];
fin.clear();
fin.close();
n++;

fin.open("27.dat");
```

APPENDIX B. CODE FOR THE CO-EVOLUTIONARY GENETIC ALGORITHM 247

```
    fin >> dat[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("28.dat");
    fin >> dat[n];
    fin.clear();
    fin.close();
    n++;

    fin.open("29.dat");
    fin >> dat[n];
    fin.clear();
    fin.close();
}
```

Appendix C

Chromosome Definitions

The following table maps the genetic code contained in the solution chromosome.

For clarification, tank rules are:

- Rule 1: Remain near friendly vehicles
- Rule 2: Move to a position to engage the enemy
- Rule 3: Move to the objective

Table C.1: Tank Physical Gene Definitions.

Positions	Definition
0-1	Engine
2-5	Gun/Ammunition Type
6-9	Ammunition Quantity(x5)
10-11	Sight
12	Autoloader
13-16	Armor Protection(x0.1m)

Table C.2: Tank Tactical Gene Definitions.

17-19	No Threat/Rule 1
20-22	No Threat/Rule 2
23-25	No Threat/Rule 3
26-28	Atk/Low Threat/Rule 1
29-31	Atk/Low Threat/Rule 2
32-34	Atk/Low Threat/Rule 3
35-37	Atk/Med Threat/Rule 1
38-40	Atk/Med Threat/Rule 2
41-43	Atk/Med Threat/Rule 3
44-46	Atk/Hi Threat/Rule 1
47-49	Atk/Hi Threat/Rule 2
50-52	Atk/Hi Threat/Rule 3
53-55	Atk/Pan Threat/Rule 1
56-58	Atk/Pan Threat/Rule 2
59-61	Atk/Pan Threat/Rule 3
62-64	Def/Low Threat/Rule 1
65-67	Def/Low Threat/Rule 2
68-70	Def/Low Threat/Rule 3
71-73	Def/Med Threat/Rule 1
74-76	Def/Med Threat/Rule 2
77-79	Def/Med Threat/Rule 3
80-82	Def/Hi Threat/Rule 1
83-85	Def/Hi Threat/Rule 2
86-88	Def/Hi Threat/Rule 3
89-91	Def/Pan Threat/Rule 1
92-94	Def/Pan Threat/Rule 2
95-97	Def/Pan Threat/Rule 3

The artillery rules are similar but include an additional rule to remain near other artillery pieces:

- Rule 1: Remain near friendly tanks
- Rule 2: Remain near friendly artillery pieces
- Rule 3: Move to a position to engage the enemy
- Rule 4: Move to the objective

Table C.3: Tank Tactical Gene Definitions (cont'd).

98-101	Fixing Force(x12.5%)
102-104	Optimal Distance(x25m)
105-107	Dollar Split(x12.5%)

Table C.4: Artillery Physical Gene Definitions.

Positions	Definition
108-109	Engine
110-111	Gun/Ammunition Type
112-115	Ammunition Quantity(x5)
116-117	Armor Protection(x0.1m)
241	Autoloader

Table C.5: Artillery Tactical Gene Definitions.

119-121	Optimal Distance
122-125	Shots/Mission
126-128	Sheaf Width(x25m)
129-131	No Threat/Rule 1
132-134	No Threat/Rule 2
135-137	No Threat/Rule 3
138-140	No Threat/Rule 4
141-143	Atk/Low Threat/Rule 1
144-146	Atk/Low Threat/Rule 2
147-149	Atk/Low Threat/Rule 3
150-152	Atk/Low Threat/Rule 4
153-155	Atk/Med Threat/Rule 1
156-158	Atk/Med Threat/Rule 2
159-161	Atk/Med Threat/Rule 3
162-164	Atk/Med Threat/Rule 4
165-167	Atk/Hi Threat/Rule 1
168-170	Atk/Hi Threat/Rule 2
171-173	Atk/Hi Threat/Rule 3
174-176	Atk/Hi Threat/Rule 4

Table C.6: Artillery Tactical Gene Definitions (cont'd).

177-179	Atk/Pan Threat/Rule 1
180-182	Atk/Pan Threat/Rule 2
183-185	Atk/Pan Threat/Rule 3
186-188	Atk/Pan Threat/Rule 4
189-191	Def/Low Threat/Rule 1
192-194	Def/Low Threat/Rule 2
195-197	Def/Low Threat/Rule 3
198-200	Def/Low Threat/Rule 4
201-203	Def/Med Threat/Rule 1
204-206	Def/Med Threat/Rule 2
207-209	Def/Med Threat/Rule 3
210-212	Def/Med Threat/Rule 4
213-215	Def/Hi Threat/Rule 1
216-218	Def/Hi Threat/Rule 2
219-221	Def/Hi Threat/Rule 3
222-224	Def/Hi Threat/Rule 4
225-227	Def/Pan Threat/Rule 1
228-230	Def/Pan Threat/Rule 2
231-233	Def/Pan Threat/Rule 3
234-236	Def/Pan Threat/Rule 4
237-240	Shooting Force(x6.25%)

Bibliography

- [1] Army Force Management School Staff AFMS, editor. *Force Integration Course Handbook*. Army Force Management School, 1996.
- [2] STEVAN JAY ANASTASOFF. Evolving mutations rates for the self-optimisation of genetic algorithms. In *Advances in Artificial Life, Proceedings from the 5th European Conference, ECAL 99*, Dario Floreano, Jean-Daniel Nicoud, and Francesco Mondad, editors, pages 74–78. Springer-Verlag, 1999.
- [3] W. BRIAN ARTHUR. *Increasing Returns and Path Dependence in the Economy*. University of Michigan Press, 1997.
- [4] W. BRIAN ARTHUR. Complexity and the economy. *Science*, 284(i5411):107–110, April 1999.
- [5] ROBERT AXTELL, ROBERT AXELROD, JOSHUA M. EPSTEIN, AND MICHAEL D. COHEN. Aligning simulation models: A case study. Technical Report 95-07-065, Santa Fe Institute, 1995.
- [6] FARHAD AZADIVAR. A tutorial on simulation optimization. In *Proceedings of the 1992 Winter Simulation Conference*, J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, editors, pages 198–204, 1992.
- [7] STEVEN C. BANKES. Tools and techniques for developing policies for complex and uncertain systems, 2000.
- [8] R. BECKERS, O.E. HOLLAND, AND J.L. DENEUBOURG. From local actions to global tasks: Stigmergy and collective robotics. In Brooks and Maes [10], pages 181–189.
- [9] ERIC BONABEAU, MARCO DORIGO, AND GUY THERAULAZ. *Swarm Intelligence*. Oxford University Press, 1999.
- [10] Rodney Brooks and Pattie Maes, editors. *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. MIT Press, 1995.
- [11] COLONEL K. STEVEN COLLIER, 2000. Army Models and Simulation Office, Office of the Deputy Undersecretary of the Army for Operations Research. From an interview with the author.

- [12] THOMAS CZERWINSKI. *Coping With the Bounds*. National Defense University Press, 1998.
- [13] Dipankar Dasgupta and Zbigniew Michalewicz, editors. *Evolutionary Algorithms in Engineering Applications*. Springer, 1997.
- [14] KEVIN DOOLEY. A complex adaptive systems model of organizational change. *Non-linear Dynamics, Psychology and Life Sciences*, 1(1):69–97, 1997.
- [15] ROBERT A. DOUGHTY. *The Breaking point: Sedan and the Fall of France*. Archon Books, 1990.
- [16] T.N. DUPUY. *Understanding War: History and Theory of Combat*. Paragon House Publishers, 1987.
- [17] CLAUS EMMECHE. *The Garden in the Machine*. Princeton University Press, 1994.
- [18] JOSHUA EPSTEIN AND ROBERT AXTELL. *Growing Artificial Societies*. Brookings Institution Press, 1996.
- [19] JOSHUA M. EPSTEIN. *The Calculus of Conventional War: Dynamic Analysis Without Lanchester Theory*. The Brookings Institute, 1985.
- [20] JOSHUA M. EPSTEIN, JOHN D. STEINBRUNER, AND MILES T. PARKER. Modeling civil violence: An agent-based computational approach. Brookings Institute, January 2001. Working paper No. 20.
- [21] THOMAS ERLENBUCH. Agent-based simulation of german peacekeeping operations for units up to platoon level. Master's thesis, Naval Postgraduate School, 2002.
- [22] R.P. FLETCHER, C. CANNINGS, AND P.G. BLACKWELL. Modeling foraging behavior of ant colonies. In *Proceedings from the 3d European Conference, ECAL 95*, F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors, pages 772–783. Springer-Verlag, 1995.
- [23] MURRAY GELL-MANN. What is complexity? *Complexity*, 1(1), 1995.
- [24] ANDREW GILL, RICHARD EGUDO, PETER J. DORTMANS, AND DION GRIEGER. Supporting the army capability development process using agent based distillations – a case study. *Journal of Battlefield Technology*, 2(3):1–6, November 1999.
- [25] JAMES GLEICK. *Genius*. Vintage Books, 1992.
- [26] C.V. GLINES. Billy mitchell, airpower visionary. *Aviation History*, 7(9), September 1997.
- [27] JEFFREY J. GOBLE. Combat assessment of non-lethal fires: The applicability of complex modeling to measure the effectiveness of information operations. Master's thesis, US Army School of Advanced Military Studies, 2002.
- [28] JAMES B. GRIER, T. GLENN BAILEY, AND JACK A. JACKSON. Response surface modeling of campaign objectives using factor analysis. *Military Operations Research*, 4(2):61–70, 1999.

- [29] PETER HAYS AND KARL MULLER. Going boldly – where? *Aerospace Power Journal*, Spring, 2001. www.airpower.maxwell.af.mil/airchronicles/apj/apj0/spr01/hays.htm.
- [30] W. DANIEL HILLIS. Co-evolving parasites improve simulated evolution as an optimization procedure. In Langton [52], pages 313–324.
- [31] JOHN HOLLAND. *Hidden Order, How Adaptation Builds Complexity*. Helix Books, 1995.
- [32] O.E. HOLLAND. Multiagent systems: Lessons from social insects and collective robotics. In *Adaptation, Coevolution and Learning in Multiagent Systems, Papers from the 1996 AAAI Symposium*, Sandip Sen, editor, SS-96-01, pages 57–62. American Association for Artificial Intelligence, 1996.
- [33] ANDREW ILACHINSKI. Irreducible semi-autonomous adaptive combat (isaac): An artificial-life approach to land warfare. Technical Report 97-61.10, Center for Naval Analysis, 1997. www.cna.org/documents/2797006110.pdf.
- [34] ANDY ILACHINSKI. Irreducible semi-autonomous adaptive combat (isaac): An artificial life approach to land combat. *Military Operations Research*, 5(3):29–46, 2000.
- [35] CLAYTON MATTHEW JOHNSON. *A Grammar-based Technique for Genetic Search and Optimization*. PhD thesis, College of William and Mary, 1996.
- [36] ARCHER JONES. *The Art of War in the Western World*. Oxford University Press, 1987.
- [37] T. JONES. A model of landscapes, 1994.
- [38] KENNETH A. DE JONG AND WILLIAM M. SPEARS. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 5(1), 1992. citeseer.nj.nec.com/dejong92formal.html.
- [39] GAL A. KAMINKA AND MILIND TAMBE. Robust agent teams via socially attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.
- [40] MINCHEOL KANG, LAURIE B. WAISEL, AND WILLIAM A. WALLACE. Team soar, a model for team decision making in simulating organizations. In *Simulating Organizations*, Michael J. Prietula, Kathleen Carley, and Less Gasser, editors. MIT Press, 1998.
- [41] STUART KAUFFMAN. *The Origins of Order : Self Organization and Selection in Evolution*. Oxford University Press, 1993.
- [42] STUART KAUFFMAN. *At Home in the Universe*. Oxford University Press, 1995.
- [43] STUART KAUFFMAN AND S. JOHNSON. Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches. In Langton [52], pages 325–369.

- [44] STUART KAUFFMAN, WILLIAM G. MACREADY, AND EMILY DICKINSON. Divide to coordinate: Coevolutionary problem solving. Technical report, Santa Fe Institute, 1994.
- [45] ORR KELLEY. *King of the Killing Zone*. WW Norton and Company, 1989.
- [46] ROBERT KEWLEY AND MARK EMBRECHTS. A computationally intelligent military tactical planning system. *IEEE Transactions on Systems, Man and Cybernetics*, 2000.
- [47] ROBERT H. KEWLEY. Automated tactical course of action development: A computational intelligence approach. Technical report, US Military Academy Operations Research Center, 2000.
- [48] JASON KINGDON AND LAURA DECKER. The shape of space. In *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms*, pages 543–548, London, 1995. IEE. citeseer.nj.nec.com/kingdon95shape.html.
- [49] KENNETH E. KINNEAR, JR. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1, pages 142–147, Orlando, Florida, USA, 27–29 1994. IEEE Press.
- [50] DONALD KNUTH. *Sorting and Searching. Volume 3: The Art of Computer Programming*. Addison-Wesley, 1973.
- [51] MACIEJ KOMOSINSKI. The world of framsticks: Simulation, evolution, interaction. In *Lecture Notes in Artificial Intelligence*, pages 214–224. Springer-Verlag, 2000.
- [52] Chris Langton, editor. *Artificial Life II, Proceedings of the Workshop on Artificial Life*. Addison-Wesley, 1991.
- [53] AVERILL M. LAW AND W. DAVID KELTON. *Simulation Modeling and Analysis*. McGraw-Hill Book Company, 1982.
- [54] JIN JOO LEE AND PAUL A. FISHWICK. Simulation-based planning for multi-agent environments. In *Proceedings of the 1997 Winter Simulation Conference*, S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, pages 405–412, 1997.
- [55] IAN O. LESSER, BRUCE HOFFMAN, JOHN ARQUILLA, DAVID F. RONFELDT, MICHELE ZANINI, AND BRIAN MICHAEL JENKINS. *Countering the New Terrorism*. Rand Corporation, 1999.
- [56] GENE LEVINSON. Crossovers generate non-random recombinants under darwinian selection. In Brooks and Maes [10], pages 90–101.
- [57] RUTH LUSCOMBE, HLEN MITCHARD, AND ANDREW GILL. Using agent-based distillations to model human intangibles for dismounted infantry combat. In *Land Warfare Conference*, Brisbane Australia, October 2002.
- [58] HUMBERTO R. MATURANA AND FRANCISCO J VARELA. *The Tree of Knowledge, Biological Roots of Human Understanding*. Shambhala Press, Boston, revised edition, 1995.

- [59] B. MCMULLIN. Computational autopoiesis: The original algorithm. Technical Report 97-01-001, Santa Fe Institute, Santa Fe, NM 87501, USA, 1997. citeseer.ist.psu.edu/mcmullin97computational.html.
- [60] MELANIE MITCHELL. *An Introduction to Genetic Algorithms*. MIT Press, 1998.
- [61] MELANIE MITCHELL, JOHN H. HOLLAND, AND STEPHANIE FORREST. When will a genetic algorithm outperform hill climbing? In *Advances in Neural Information Processing Systems 6*, J. D. Cowan, G. Tesauero, and J. Ispector, editors. Morgan Kaufmann, 2002.
- [62] Modsaf 5.0 functional description document. United States Army Simulation, Training and Instrumentation Command, January 1999. Prepared by the Science Applications International Corporation and Lockheed Martin Informations Systems, www.modsaf.org/publicmodsaf1.html.
- [63] KAI NAGEL AND STEEN RASMUSSEN. Traffic at the edge of chaos. In Brooks and Maes [10], pages 222–235.
- [64] JAN PAREDIS. Coevolving cellular automata: Be aware of the red queen! In *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA97)*, T. Baeck, editor. Morgan Kaufmann, 1997.
- [65] STEVE PARK AND LARRY LEEMIS. *Discrete Event Simulation: A First Course*. unpublished, 2000.
- [66] CARLOS-ANDRES PENA-REYES AND MOSHE SIPPER. Fuzzy CoCo: Balancing accuracy and interpretability of fuzzy models by means of coevolution, 2002. [url=citeseer.nj.nec.com/501020.html](http://citeseer.nj.nec.com/501020.html).
- [67] PETER P. PERLA. *The Art of Wargaming: A Guide for Professionals and Hobbyists*. Naval Institute Press, 1990.
- [68] ELENA POPOVICI. Understanding landscapes. Personal communication with the author.
- [69] THOMAS RAY. Evolution, ecology and optimization of digital organisms. Santa Fe Institute Working Papers.
- [70] THOMAS RAY. An approach to the synthesis of life. In Langton [52], pages 371–408.
- [71] GENERAL DENNIS REIMER. The army after next: Knowledge, speed and power. *Military Review*, pages 2–7, 1999.
- [72] MITCHELL RESNICK. *Turtles, Termites, and Traffic Jams*. MIT Press, 1994.
- [73] CRAIG REYNOLDS. Steering behaviors for autonomous characters. www.red3d.com/cwr/steer/gdc99/index.html.
- [74] CRAIG W. REYNOLDS. Competition, coevolution and the game of tag. In Brooks and Maes [10], pages 59–69.

- [75] FIELD MARSHALL ERWIN ROMMEL. *Attacks*. Athena Press, 1979. Reprinted from the 1937 book by the same title.
- [76] MAJOR GENERAL ROBERT H. SCALES. *Future Warfare*. US Army War College, Carlisle Barracks, PA, 1999.
- [77] LEE SCHRUBEN. Simulation optimization using frequency domain methods. In *Proceedings of the 1986 Winter Simulation Conference*, J. Wilson, J. Henrickson, and S. Roberts, editors, pages 366–369, 1986.
- [78] KARL SIMS. Evolving 3d morphology and behavior by competition. In Brooks and Maes [10], pages 28–39.
- [79] HOKKY SITUNGKIR. Money-scape: A generic agent-based model of corruption. Working Papers, 2004. Economic Working Papers Archive, Washington University, Saint Louis, <http://econpapers.hhs.se/paper/wpawuwpc0/0405008.htm>.
- [80] WILLIAM M. SPEARS. Adapting crossover in evolutionary algorithms. In *Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384. MIT Press, 1995.
- [81] WILLIAM M. SPEARS AND KENNETH A. DEJONG. An analysis of multi-point crossover. In *Proceedings of the Foundation of Genetic Algorithms Workshop*, pages 301–315, 1990. citeseer.nj.nec.com/97440.html.
- [82] LUC STEELS. *The origins of intelligence*, 1996.
- [83] TARJA SUSI. Social cognition, stigmergy, and artefacts[sic]: A comparative analysis of theoretical frameworks for the study of computer-mediated collaborative activity. Master's thesis, University of Skovde, Sweden, 2000.
- [84] MILIND TAMBE, JAFAR ADIBI, YASER AL-ONAIZAN, ALI ERDEM, GAL A. KAMINKA STACY MARSELLA, AND ION MUSLEA. Building agent teams using an explicit teamwork model and learning. *Artificial Intelligence*, 110:215–239, Feb 1999.
- [85] KURT THEARLING AND THOMAS S. RAY. Evolving parallel computation. *Complex Systems*, 10, 1997. www3.shore.net/kht/text/evpar/evpar.shtml.
- [86] YUKIHIKO TOQUENAGA, ISAMU KAJITANI, AND TSUTOMU HOSHIMO. Egrets of a feather flock together. In Brooks and Maes [10], pages 140–151.
- [87] JEFFREY VENTRELLA. Designing emergence in animated artificial life worlds. In *Virtual Worlds 98, Lecture Notes in Artificial Intelligence*, pages 143–155. Springer-Verlag, 1999.
- [88] M. S. VOSS. Complex adaptive systems + soft computing = emergent design systems. In *ASC 2000: Third IASTED International Conference on Artificial Intelligence and Soft Computing*, Banff, Alberta, 2000.
- [89] DANIEL S. WELD. *Theories of Comparative Analysis*. MIT Press, 1990.

- [90] TONY WHITE, BERNARD PAGUREK, AND FRANZ OPPACHER. (asga): Improving the ant system by integration with genetic algorithms. In *Genetic Programming 1998: Proceedings from the Third Annual Conference*, John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors, pages 610–617, University of Wisconsin, Madison, Wisconsin, USA, 1998. Morgan Kaufmann.
- [91] A. MARTIN WILDBERGER. Introduction and overview of artificial life, evolving intelligent agents for modeling and simulation. In *Proceedings of the 1996 Winter Simulation Conference*, J.M. Charnes, D.J. Morrice, D.T. Brunner, and J.J. Swain, editors, 1996.
- [92] RONALD F. A. WOODAMAN. Agent-based simulation of military operations other than war small unit combat. Master's thesis, Naval Postgraduate School, 2000.

VITA

Steven Mains

Colonel Steven Mains was born in Downey, California on December 2, 1958. He graduated from the US Military Academy in 1980 with a BS in Arabic and Civil Engineering. He received a MS in Systems Management from the University of Denver in 1990 and an MS in Computational Operations Research from the College of William and Mary in 2000.

During his 24 years of active service as a U.S. Army Armor officer, he has served in Cavalry and Armor units in Germany and Fort Hood. His operational assignments include service in the 11th Armored Cavalry Regiment patrolling the Inter-German Border and the 1st Armored Division in Operation Desert Storm. A member of the original JFCOM Lesson Learned Team, he led the Coalition Force Land Component Command Team deployed to Kuwait and Iraq during Operation Iraqi Freedom. Subsequently, he has deployed to collect Lessons Learned in Afghanistan and Kosovo.

His military education includes the Armor Basic and Advanced Courses, Ranger and Airborne Schools, and the British Army Command and Staff College.

His military awards include the Defense Superior Service Medal, Legion of Merit, Bronze Star, and Valorous Unit Award. His campaign medals include the Southwest Asia Service Medal with two Battle Stars.