2006

Modeling Autonomous Agents In Military Simulations

Varol Kaptan
University of Central Florida

MODELING AUTONOMOUS AGENTS IN MILITARY SIMULATIONS

by

VAROL KAPTAN
B.S. Middle East Technical University, 1996
M.S. Middle East Technical University, 1999

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2006

Major Professor:
Erol Gelenbe

ABSTRACT

Simulation is an important tool for prediction and assessment of the behavior of complex systems and situations. The importance of simulation has increased tremendously during the last few decades, mainly because the rapid pace of development in the field of electronics has turned the computer from a costly and obscure piece of equipment to a cheap ubiquitous tool which is now an integral part of our daily lives. While such technological improvements make it easier to analyze well-understood deterministic systems, increase in speed and storage capacity alone are not enough when simulating situations where human beings and their behavior are an integral part of the system being studied. The problem with simulation of intelligent entities is that intelligence is still not well understood and it seems that the field of Artificial Intelligence (AI) has a long way to go before we get computers to think like humans.

Behavior-based agent modeling has been proposed in mid-80's as one of the alternatives to the classical AI approach. While used mainly for the control of specialized robotic vehicles with very specific sensory capabilities and limited intelligence, we believe that a behavior-based approach to modeling generic autonomous agents in complex environments can provide promising results. To this end, we are investigating a behavior-based model for controlling groups of collaborating and competing agents in a geographic terrain.

In this thesis, we are focusing on scenarios of military nature, where agents can move within the environment and adversaries can eliminate each other through use of weapons. Different aspects of agent behavior like navigation to a goal or staying in group formation, are implemented by distinct behavior modules and the final observed behavior for each agent is an emergent property of the combination of simple behaviors and their interaction with the environment. Our experiments show that while such an approach is quite efficient in terms of computational power, it has some major drawbacks.

One of the problems is that reactive behavior-based navigation algorithms are not well suited for environments with complex mobility constraints where they tend to perform much worse than proper path planning. This problem represents an important research question, especially when it is considered that most of the modern military conflicts and operations occur in urban environments. One of the contributions of this thesis is a novel approach to reactive navigation where goals and terrain information are fused based on the idea of transforming a terrain with obstacles into a virtual obstacle-free terrain. Experimental results show that our approach can successfully combine the low run-time computational complexity of reactive methods with the high success rates of classical path planning.

Another interesting research problem is how to deal with the unpredictable nature of emergent behavior. It is not uncommon to have situations where an outcome diverges significantly from the intended behavior of the agents due to highly complex nonlinear interactions with other agents or the environment itself. Chances of devising a formal way to predict and avoid such abnormalities are slim at best, mostly because such complex systems tend to be be chaotic in nature. Instead, we focus on detection of deviations through tracking group behavior which is a key component of the total situation awareness capability required by modern technology-oriented and network-centric warfare. We have designed a simple and efficient clustering algorithm for tracking of groups of agent suitable for both spatial and behavioral domain. We also show how to detect certain events of interest based on a temporal analysis of the evolution of discovered clusters.

To my family

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor, Professor Erol Gelenbe, for his encouragement and for the guidance he provided during my research. This dissertation would not have been possible without his constant support. I would also like to thank the members of my examining committee, Professor Ratan Guha, Professor Lesia Crumpton-Young and Professor Annie Wu for their feedback and help.

I am also indebted to my fellow graduate students, as well as the faculty and staff of the School of Computer Science for providing environment and resources which made this effort possible.

This work was partially supported by U.S. Army STRICOM through contracts N61339-02-C0050 and N61339-02-C0080, the Data and Information Fusion DTC by a contract from General Dynamics UK to Imperial College under Project 6.8 (Future Data Fusion Systems Design and Demonstration, and by the MoD Defence Science and Technology Laboratory.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Problem Statement

The main goal of this thesis is to investigate the problem of modeling autonomous agents within the setting of military conflicts. In particular, we will concentrate on multi-agent systems in which teams of agents embedded in a terrain are able to cooperate and compete with each other. We will further limit our focus on models which can be employed as training and assessment tools in augmented-reality simulations which impose hard real-time constraints on agent acting in unpredictable and complex environments.

The main argument we put forward is that research into behavior-based agent models can provide significant benefits towards achieving these goals. In support of this statement, we show how a particular behavior-based agent model can be used as a valuable tool for modeling a variety of situations arising in military conflicts. In addition, we provide novel contributions to two specific problems (i.e. navigation in complex environments, and detection and tracking of groups) which are of particular interest in such models.

1.2 Thesis Organization

The rest of this chapter provides discussions on fields related to our main problem. We examine the role of Simulation as a training and situation assessment tool, Augmented Reality within the context of training, provide background on Classical and Behavior-based Artificial Intelligence as tools for Autonomous Agent Control, and the fundamental role that Data and Information Fusion plays in the subject of modern military conflicts.

Chapter 2 describes in broad terms the behavior-based agent model that we propose with particular focus on the types of behaviors that need to be employed for modeling the range of events and capabilities associated with military scenarios.

Chapter 3 presents a detailed description of the architecture of the simulation testbed that we have implemented. In particular, it describes the agent environment, how different behaviors are implemented and combined, and a number of metrics related to the performance of agents.

Chapter 4 includes a detailed description of three sets of experiments and discussions of the respective results. The first set of experiments provides evidence of how behavior combination can improve the overall success of agents. The following two sets of experiments are designed to show how our approach can be used to provide statistically significant answers to questions of interest and how our system can be extended in mission-specific ways. As part of the discussion, we also identify two important areas of further research. These are the problem of navigation in complex environments and the problem of detection and tracking groups of agents. We address these problems in Chapter 5, where we describe a novel algorithm for reactive navigation and in Chapter 6, where we describe algorithms for group detection and for tracking the evolution of dynamic groups for the purpose of detecting events of interest.

Chapter 7 provides a summary of our contributions and a discussion of the limitations of our approach as well as possible direction for further research. The Appendix includes descriptions of the Random Neural Network(RNN), Reinforcement Learning in RNN and the detailed results of the first experiment described in Chapter 4. Some of the material presented in this thesis has been included in published journal papers and presented at international conferences. This includes material in Chapters 2 and 3 [GHK05, GKH04, GKW05, GW06], Chapter 4 [GKH04, GKW04, GKW05], Chapter 5 [KG06] and Chapter 6 [GHK05, GKH04, GKW06].

1.3 Role of Simulation in Training and Situation Assessment

One of the increasingly important application areas of simulation is in education and training, where simulation can be used to illustrate concepts and provide exercises that allow the learner to train in a realistic environment. The use of real scenarios enhanced by “what if” situations can offers a very stimulating learning setting for self-learning and self-evaluation.

The main advantage of using simulations in training is that they can provide significant reductions to costs and hazards. This is particularly important in the field of military training systems where traditional exercises have costs and hazard levels which approach those of military operations [Ken99, Sch04]. Another application area where simulations can be very valuable is the evaluation of alternative plans of actions related to a system or situation where real experimentation is not possible or prohibitively expensive.

These advantages, however, can be delivered only when a simulation models the system in question to an acceptable level of accuracy and realism. For example, the use of purely synthetic scenarios in military training systems can significantly reduce the realism of an exercise, and therefore have a negative effect on the learning process of a trainee and the quality of the acquired experience. On the other hand, using an Augmented Reality (AR) system to insert simulation driven virtual objects in real scenes can offer a higher degree of motivation to the learner, who will face a realistic stimulus approaching that of a real situation under real-time operating constraints.

1.4 Augmented Reality

Augmented Reality refers to the process of enhancing one's perception of the real world by augmenting a mostly real-word environment with some computer-generated sensory information. There are many application areas of AR, however we will limit our discussion to the subject of embedded training, where the term *embedded* means that the training system is built into an actual operational system, and that the operational system and the training system are designed so that they can be used jointly. Many fields of application for augmented reality based training systems have a need for real-time interaction between the learner and the augmented reality which is being observed.

The ultimate goal of augmented reality is to include a significant human sensory environment with a visual component, as well as sound, touch, physical motion and pressure, and even smell. Thus, an augmented reality surgical training operation table, could allow the surgeon to sense the smell of blood and of the chemical products which are being used, as well as to feel the pressure of the organs as the synthetic surgical instruments are being applied to the synthetic patient, whose resulting vital signs and endoscopic images are also being shown on an appropriate set of screens. However, due mostly due to limitations of existing technology, modern AR systems operate almost exclusively on the visual component of perception.

A visual augmented reality system creates a combination of a real and virtual scene in which the user perceives a significant difference between the real and augmented world. One of the difficult technical issues in augmented reality is the *registration* problem, which refers to the need for determining the isomorphism between objects and features in a live scene with the corresponding features and the corresponding objects in an augmented version of that scene. Errors in registration will generate visual inconsistencies between real and virtual images with obvious consequences on the value of the augmented reality system for simulation purposes.

This issue has been addressed widely and it is well known that registration using only information from a sensor based tracking system cannot achieve a perfect match [BN95]. Most of the approaches to robust registration have to combine tracking sensor input with some image processing algorithms in order to improve registration.

One approach is to detect features in the real image and use them to enforce registration. Another approach is to place special marks (e.g. LEDs [BN95], circles [Mel95], a calibration grid [LP98]) in the environment. Image-processing algorithms detect the locations of these marks and use them to enforce registration, assuming that one or more special marks are visible at all times. The requirement for placing dedicated marks is somewhat limiting to the applicability of this method to well-controlled environments.

Yet another approach [Mel95] uses a survey of the the live environment with real-time instrumentation, providing more information about objects and their distances in the live environment, but requires specific equipment and significant amounts of additional computation for the interpretation of the sensors' output.

Almost all augmented reality techniques assume that virtual objects and live objects have the same detailed shape. This assumption is only valid for objects with well-defined shapes such as roads and buildings: many virtual object generators will use a simplified representation, and will even sometimes only make use of templates; e.g., a synthetic pine tree may be some idealized template of a pine tree, rather than the actual pine tree being represented at some location.

1.5 Controlling Autonomous Agents

Although research in augmented reality is focused predominantly on seamless integration of synthetic and real components, in the context of a training system, the behavior of injected artificial entities can have impact on the effectiveness of the training which is as important as the effect of visual realism. This is especially important in the context of simulations designed for training personnel or evaluating a “what if ...” situation. In such simulations, the behavior of agents will have an important effect on the final outcome in the form of acquired training experience. Unrealistic agent behavior, e.g., in the form of very limited or even extremely advanced intelligence may result in poor performance of the trainees in a real-life situation.

Agent behavior in a sophisticated simulated environment can be very complex and may involve many entities. Intelligence can be employed at very different levels. A very simple example will be an agent that has to go from one position to another trying to minimize travel time. A very complex example of intelligent behavior can include the decision to cancel the mission of a group of entities and relocating them as a backup for another group. While the first problem can be easily solved by a single autonomous entity, the second will involve some authority that can make a higher-level decision based at least partially on feedback from the lower-level autonomous agents or sensors tracking them. In other words, a solution to the latter problem requires the ability to acquire some kind of (at least) partial situation awareness - a capability usually associated with high-level data and information fusion.

Multi-Agent systems are an important field in AI since they emerge as a natural way of dealing with problems of distributed nature. Such problems exist in a diversity of areas like military training, games and entertainment industry, management, transportation, information retrieval and many others.

Multi-agent systems interacting with the real world face some fundamental restrictions. Some of these are:

1. They have to deal with an unknown and dynamic environment
2. Their environment is inherently very complex
3. They have to act within the time frame of the real world
4. The level of their performance should be "believable" to real people when agents are used to simulate human behavior.

Ideally, for meeting these requirements agents have to be able to learn, coordinate and collaborate with each other as well as people do. In the context of augmented reality simulations, this means dealing with a very complex environment, incomplete information and hard timing constraints – properties which make learning a very difficult task.

The main problem with learning in multi-agent systems is the "exponential explosion" in the problem state-space. The usual approach of functional decomposition of the state-space can offer some improvement but it usually very limited. Even simple multi-agent systems consisting of only a few agents within trivial "toy" environments can quickly approach prohibitively expensive computational requirements related to learning [Tan93, OF96, OF97]. The extensive amount of time and computational resources required for a usual learning process is another obstacle which is important in augmented reality systems.

1.5.1 Classical AI

The field of Artificial Intelligence (AI) deals with the problem of creating machines which can exhibit intelligence comparable to or even better than the natural intelligence of biological organisms and humans in particular. It is generally accepted that AI was born as a distinct field of research in 1955 during a summer research conference organized in Dartmouth College.

In the first few decades, AI researchers focused mainly on systematic approaches based on the assumption that an intelligent machine can build an internal representation of its environment through acquiring information and then act in an intelligent way by performing planning over this abstract representation. It is now generally accepted that this *classical* approach was too optimistic and naive, and has largely failed to deliver the results it initially promised. However, this fact does not diminish the importance of classical AI research which has provided many valuable contributions to the field of Computer Science and Engineering, and many of these contributions are widely used on a daily basis in various industrial and consumer products.

The following observations may help understand the reasons for this failure. First, it is a well-known fact that there is a great disparity between the computational model and capacity of modern computers and biological systems. Biological computational systems are built as a complex inter-connected network of a large amount of neurons which process information in an extremely parallel fashion and this network is amazingly redundant and robust to failure of its components. Modern computers, on the other hand, are largely sequential information processors which are still far away from achieving the computational power of natural systems. However, the architectural discrepancy and difference in raw computational power between natural and artificial intelligent systems are not significant problems. The real big issue facing AI is the fact that natural intelligence is still a poorly understood phenomena and as a result, computational power alone is generally not considered sufficient for replicating natural intelligence (for example, see [Pen89, Teg00, HHT02] for a debate on whether conscience can be achieved with classical computers/physics and [Koc97] for discussions on how information is possibly transmitted in biological neural networks).

1.5.2 Behavior-based AI

Behavior-based AI emerged as a school of thought as a direct result of the failure of *classical* AI to deliver. The main conjecture of Behavior-based AI is that natural intelligence is not necessarily a result of a well-defined algorithmic model of information flow from perception through cognition to action, but rather emerges in a non-trivial way from the complex interactions between tightly coupled simple behavioral components.

In this thesis, we will use the term *emergent* to describe properties of agents or aspects of agent behavior which form as a result of such complex interactions in ways which make it impossible (either theoretically or practically) to predict them or control their effects by means of a simple analysis of the constituent phenomena responsible for their manifestation. As a result of this unpredictability, emergence will sometime produce behavior which is unexpected or undesired. We will, therefore, use the same term to describe unexpected or undesirable situations [1] when they occur as a result of emergence in order to emphasize the irreducible nature of such phenomena.

Behavior-based AI was first popularized by the work of Brooks [Bro86]. It is based on using simple behavior patterns as basic building blocks and trying to implement and understand intelligent behavior through the construction of artificial life systems. His inspiration comes from the

[1] Chapter 6, for example, is dedicated to detection of such events.

way intelligent behavior emerges in natural systems studied by Biology and Sociology. Brooks proposes to combine behaviors through a vertical hierarchy of simple computational elements where higher modules can suppress the outputs of lower modules and replace their functionality when necessary. This particular model is called “subsumption architecture”.

The main reason for the recent interest in behavior-based systems is that they have been more successful in modeling simple intelligence than classical approaches. In a series of papers, Brooks [Bro99] describes a number of robots built on the subsumption architecture. These robots operate in the real world and show a level of intelligence reminiscent to that of simple insects.

Another example of a biology-inspired model of behavior is the “boids” architecture proposed by Reynolds [Rey87]. He uses a set of simple rules in order to model the coordinated motion of animal groups such as bird flocks and fish schools. These rules are *separation* (steer to avoid crowding local flockmates), *alignment* (steer towards the average heading of local flockmates) and *cohesion* (steer to move toward the average position of local flockmates).

A similar approach for modeling group behavior inspired by physics rather than biology, is the Social Potential Fields method [RW99]. It is based on the observation that combination of attractive and repulsive forces can create a variety of group formations between particles in physics.

Because of their performance and ability to scale to a large number of agents, these methods of modeling groups have been widely deployed in technology-driven fields such as the computer-games industry [Rey99, Pot99].

One of the main problems of behavior-based systems is the question of how to combine different (possibly conflicting) behaviors in order to achieve a desired outcome. For example, in the original subsumption architecture described by Brooks, this problem is solved by a manual wiring scheme in the form of an explicit network of suppression and inhibition relations between computational elements. Another example of a manually-generated approach for combining behaviors is based on fuzzy logic [SRK99]. While such approaches can be useful for creating simple robots, their "hand-crafted" nature limits their ability to scale to a large number of behaviors and makes them unsuitable for designing complex agents. More flexible examples of behavior coordination include Action Selection Methods [Mae90] where behaviors are selected based on their activation levels which are a function of agent goals and incoming sensory information or the behavior arbitration scheme of the DAMN architecture [RT95] where actions are selected based on the votes of behaviors.

Ideally, a better way to deal with the problem of behavior combination would be to let a system learn how to combine behaviors in an unsupervised manner (possibly including contextual information as well) by using techniques like reinforcement learning in the behavior domain [Mat97, Bal98]. Unfortunately, such methods suffer from an exponential explosion in complexity when applied to multi-agent systems and are of little practical value in real life.

Good discussions on the historical development of behavior-based AI can be found in [Ste93, Bro99] and an extensive treatment of the subject is given in the book of Ronald Arkin [Ark98].

1.6 Data and Information Fusion

The field of Data Fusion can be described as the study of how to integrate data from multiple sources/sensors for improving the performance of algorithms and systems [HL01]. Although its potential application area is quite large, the emergence of Data Fusion as a separate field and the bulk of current work is driven mostly by interest from the military research community. The cause of this interest is related to the fact that recent advances in information and communication technology has changed the nature of modern warfare from platform-centric to network-centric. Traditionally, military information gathering and processing has been a job for human experts. However, new sensor processing techniques, weapon platforms and communication systems may require decision makers to deal with excessively large amounts of data and possibly act within a very short time. The availability of tools and technologies for automatic identification and tracking of targets, automatic thread recognition, guidance for autonomous vehicles and smart weapons, and building a coherent picture of the battlefield situation is considered a key success factor for commanders on the field [CG98].

The advantages of fusing data from multiple sensors can be illustrated as follows:

- Combining a number of measurements from a sensor (or identical sensors) can improve the statistical quality of the estimate of an observable parameter of a tracked entity

- Multiple sensor readings can be used in order to infer parameters which a single sensor cannot provide (for example, triangulation of an entity’s position from multiple sensor readings)

- Combining readings from different sensors can dramatically improve estimates of parameters (for example, combination of readings from sensors with high range accuracy and high angular resolution)

- Low-level tracking information can be used in order to infer/discover relationship between entities, their organizational structure, their interaction with the environments and their intentions

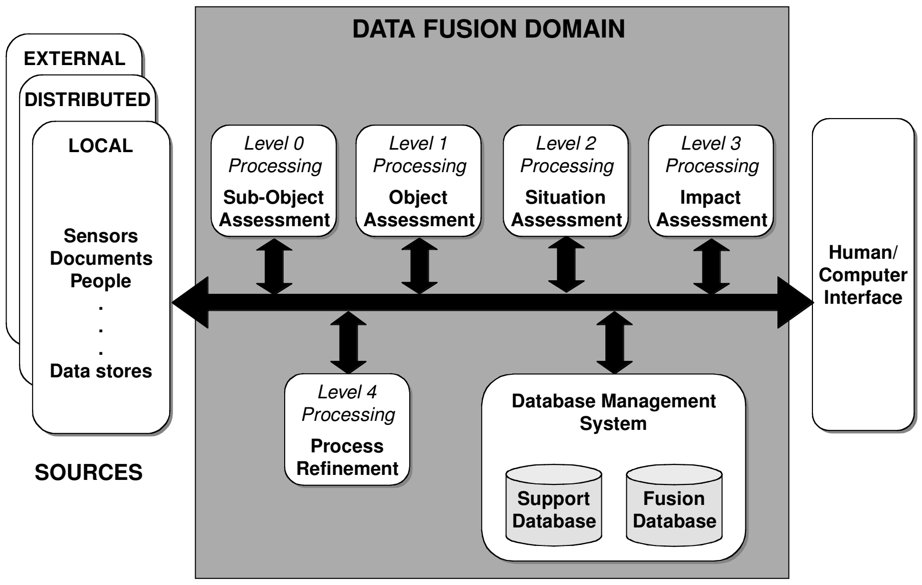


Figure 1.1: The JDL Data Fusion Model (1998 revision)

It is easy at this point to see that data fusion can be employed to data of very diverse characteristics. For example, the first three cases in the list above are usually considered to be primarily engineering problems that can be solved with the right technology. The last case, on the other hand, cannot be solved though engineering alone but will require a more human-centric skills, including extensive military experience, insight into human behavior and strong tactical planning and reasoning skills.

One of the side effects of the interdisciplinary nature of data fusion is the difficulty of cross-pollination between application-specific boundaries during the early years of the development of the field. The U.S. Joint Directors of Laboratories (JDL) data fusion Working Group was formed in 1986 in order to provide a common foundation and develop a standard data fusion terminology and lexicon. One of the contributions of the JDL has been the construction of a functional model of data fusion. The model has since then been revised by JDL as well as other entities [SB04, LBR04]

- the 1998 JDL revision is shown in Figure 1.1. The JDL model differentiates functions into fusion levels related to refinement of signals, objects, situations, threats and processes [HL01]. These can be described as follows:

- Level 0 – Signal (Sub-Object) Assessment – estimation and prediction of signal-level states based on sensor measurements
- Level 1 – Object assessment – estimation and prediction of object states on the basis of inferences from observations
- Level 2 – Situation Assessment – estimation and prediction of object states based on inferred relations between them
- Level 3 – Threat (Impact) Assessment – estimation and prediction of effects of predicted actions by the objects involved
- Level 4 – Process Refinement – adapting data acquisition and processing in order to support mission objectives

Some researchers prefer to differentiate between the type of data being processed at different levels by using the terms data, information [2] and knowledge to underline the change in nature of processing from lower levels to higher ones. This practice, however, is not universally accepted and can be even considered controversial in some circles.

As mentioned earlier, while lower-level fusion tasks are mostly engineering challenges, the higher-level fusion require more intelligence. When an autonomous system implementation based

[2]hence the term information fusion

on modern computing technology is considered, this implies the requirement of an artificial intelligence with performance levels comparable to human experts. Because of this natural connection between data fusion and military conflict simulations, we are using data fusion terminology whenever possible in order to emphasize the multi-disciplinary impact of our results and put our contributions in proper context.

CHAPTER 2

A BEHAVIOR-BASED AGENT MODEL

2.1 Introduction

The focus of this thesis is the modeling of multi-agent systems in the context of real-time simulation of military scenarios. As a direct consequence of this constraint, the design of our agent model and operating environment has been strongly influenced by the particular characteristics that are of primary importance in such situations. The purpose of this chapter is to provide a broad definition of a behavior-based agent model and a description of the types of individual behaviors we think are representative of the different aspects of action required to successfully model military scenarios.

2.2 Behavior Types

The approach we propose is based on the idea of behavior-based robotics where the overall actions of an agent are emerging as a result of the combination of simple behaviors which model different mechanisms involved in the decision making process. We will broadly classify these behaviors as being related to *navigation, grouping, adversarial action* and *other actions*:

- Navigation behaviors are responsible for moving agents between different locations in the terrain.

- Grouping behaviors are helping agents create and maintain spatial formations while performing missions.

- Adversarial behaviors model interactions between enemy forces.
- Other behaviors include actions which do not directly fit into any of the above categories.

The emergence of an agent decision can be described by the following equation:

$$D = aD_n \oplus bD_g \oplus cD_a \oplus dD_o$$

where D describes the individual's overall action, a, b, c and d are weighting parameters and:

- D_n represents the individual navigation decisions,
- D_g represents the grouping-related decisions,
- D_a represents the adversarial decisions,
- D_o represents other decisions,
- $\oplus$ is an operation which combines the separate decisions into a physical action.

The weighting parameters can be used to influence which aspects of agent behavior are more important in different situations. We separate behaviors in such classes with the hope that by combining methods which specialize in solving particular parts of a mission objective, we can provide a model of multi-agent control which is computationally efficient and at the same time expressive enough to be of practical significance. The following sections describe the constituent behaviors in detail.

2.2.1 Navigation Behaviors

The goal of navigation behaviors is to move from an initial location in the terrain to a final destination. This problem can be formalized as a goal function G which agents are trying to minimize. A very simple example is the case when G represents the current distance of an agent to its destination. A more complex example is when an agent is trying to minimize the expected time of arrival to a destination while navigating in a dangerous environment. Suppose that the agent has to get to a destination d as quickly as possible. If the agent is hit, it will have to be replaced by another agent, effectively increasing the time to complete the mission. The effect of taking a decision D of an agent located at position x can be modeled with the following equation:

$$G_D(x,d) = s^{-1} + (1 - p(x + sD))G(x + sD, d) + p(x + sD)(T + G(0,d))$$

where

- x is the current position,
- d is the final destination,
- s is the speed of the agent (s^{-1} is the time for one step motion),
- $p(x)$ is the probability of being hit at position x,
- D is the directional decision of the agent,
- T is the cost of restarting the mission,
- $G(u,v)$ is the cumulative cost incurred to get from u to v.

One of the challenges is finding $G(u, v)$. Ultimately, it concerns the future performance of the agent strategy and may not be possible to compute, especially when the agent operates in a dynamic environment. Fortunately, $G(u, v)$ can be estimated based on current sensory information and previous experience in order to allow for real-time decision making.

One of the methods that we investigate is a neural network based decision process where the goal function is estimated through reinforcement learning applied to the Random Neural Network [GSX01]. Another method, which we discuss in detail in chapter 5, is based on applying novel data fusion techniques for augmenting simple reactive navigation with terrain information.

One of the main problems of quantitative formulations of goal-based actions is that they require numerical expression of goals rather than linguistic or logic. On the other hand, such an approach offers an easy infrastructure for efficient computation which is important in a real-time environment. Numerical expression of goals can also offer an efficient benchmark for measuring success during and after the completion of a mission.

2.2.2 Grouping Behavior

We use Grouping behaviors for representing the formation of spatial structures of agents. Our model is based on the idea of Social Potential Fields (SPF) proposed by Reif et. al. [RW99]. This is a simple distributed-control approach inspired by the attractive and repulsive forces between charged particle in physics. Their work was inspired by the artificial potential field method pioneered by Khatib [Kha86]. The boids approach proposed by Reynolds [Rey87] is functionally similar to the potential methods, although it is inspired by biology rather than physics and has therefore a somewhat different formulation.

In its most general form, a SPF consist of a number of attractive and repulsive forces between agents. The social force laws can differ from physical forces in the sense that they are not necessarily symmetric and can be dynamically changed. There have been attempts to use SPFs as a complete self-contained way of modeling action, but results have not been promising. It is for this reason that we will use SPFs primarily for modeling grouping behavior, for which they happen to be particularly well suited. For example, a combination of a long-range attractive force dominated by a short-range repulsive force can be used to keep a group agents in a particular spatial formation.

Although traditionally SPF forces are defined as a function of inter-agent distances, it may be interesting to see how they can be used to model group behavior based on relations other than spatial. The main advantage of the SPF formalism is that it offers a uniform, simple and efficient computational mechanism for forming groups of agents.

2.2.3 Adversarial Behaviors

The purpose of adversarial behaviors is to act in a manner that negatively affects the performance of agents which are considered enemies. There are various ways in which such effects can be achieved in a military situation. For example,

1. Weapons can be used for the purpose of eliminating enemy agents or other infrastructure.
2. Weapons can also be used as a deterrent in order to discourage certain enemy actions.
3. Disruption of communications or the act of providing false information can be used to affect the decision process of an enemy.

4. Limiting access to resources or cutting supply routes can be used to diminish enemy capabilities.

As it can be seen from these few examples, there are many ways in which agents can employ adversarial actions and a comprehensive treatment of this subject is well-beyond the scope of our research. Moreover, proper modeling of such actions usually requires access to restricted technical and operational know-how.

Fortunately, in the context of the military scenarios that we are considering, adversarial action is usually implemented through use of weapons. We will, therefore, focus primarily on modeling adversarial action in the case of using a weapon to eliminate enemy agents.

2.2.4 Other Behaviors

Some aspects of agent behavior may not be related directly to adversarial action, motion towards a destination or creating spatial social structures. Such behaviors may be employed for the purpose of supporting goals of secondary nature or in order to provide incremental improvements of a mission outcome.

For example, an agent in a group may decide to imitate the decisions of other members of the group. The imitation decision can be formed by taking the average of the observed decisions of other group members. This process can be useful when a quick decision is required due to time constraints. Imitation can also be useful for harnessing the experience of more successful agents by their peers as a way of mitigating lack of experience or knowledge.

Another example is the case when an agent decides to take a course of action which is distinctively different from the average behavior of its group. This behavior can be used used to model aspects of human behavior related to fatigue, poor judgment or insubordination.

A third example is the action of obstacle avoidance. Instead of trying to achieve a primary objective, this behavior plays a supporting role in the process of navigation in a terrain with mobility constraints.

2.3 Combining Behaviors

Instead of providing a well-defined methods of combining behaviors, our agent model is based on the observation that combination of simpler actions can be considered as just another behavior. There are two main reasons for choosing such an approach. First, this architecture allows us to choose a method of combination which is as simple as necessary for a particular purpose. In fact, the implementation described in the next chapter shows that even very simple combination methods can provide promising results. Second, this decision makes our behavior model very flexible by allowing incorporation of mission-specific capabilities only as needed.

2.4 Measuring Performance

One of the advantages of having a decision process based on the formalism described above is that it can allow a rich set of quantitative metrics for measuring agent performance. Besides giving us an idea of how successful the proposed model is in solving the problems addressed in a simulation, the performance measurements can also be used to actually improve the behavior of the agents by providing them with an efficient feedback mechanism. Such a feedback can be used to facilitate a process of learning better behavior, both during a mission and between different missions.

CHAPTER 3

TESTBED IMPLEMENTATION

3.1 Introduction

Two separate systems have been designed and implemented as part of our research. The first system is an Augmented Reality (AR) simulator which can inject virtual moving objects in a video recording of a real environment. The second system is an Autonomous Agents (AA) simulator. The latter is the testbed on which we analyze and test behavior-based approaches to autonomous agent control and it is the main focus of this thesis.

The systems are designed to operate separately but there is also a glue layer which allows them to operate together - i.e. the Autonomous Agents testbed can control the virtual agents in the Augmented Reality simulator. There are two main reasons for this separation:

1. It is much easier to develop the AR system on a table-top or indoor environment. An outdoor testbed would have required more expensive equipment (vehicle-mounted cameras, GPS receivers and accurate topological maps of an area, for example) and dedicated vehicles/agents. Besides the extra cost, an outdoor testbed would also be probably more dangerous to operate.

2. The ability to detach the AA testbed from the AR system enables the standalone simulation of agents embedded in environments with very large dimensions. It also allows operation in time scales that are much faster than real time. Such a capability is essential for performing comprehensive analysis of behaviors over a large number of simulation scenarios and

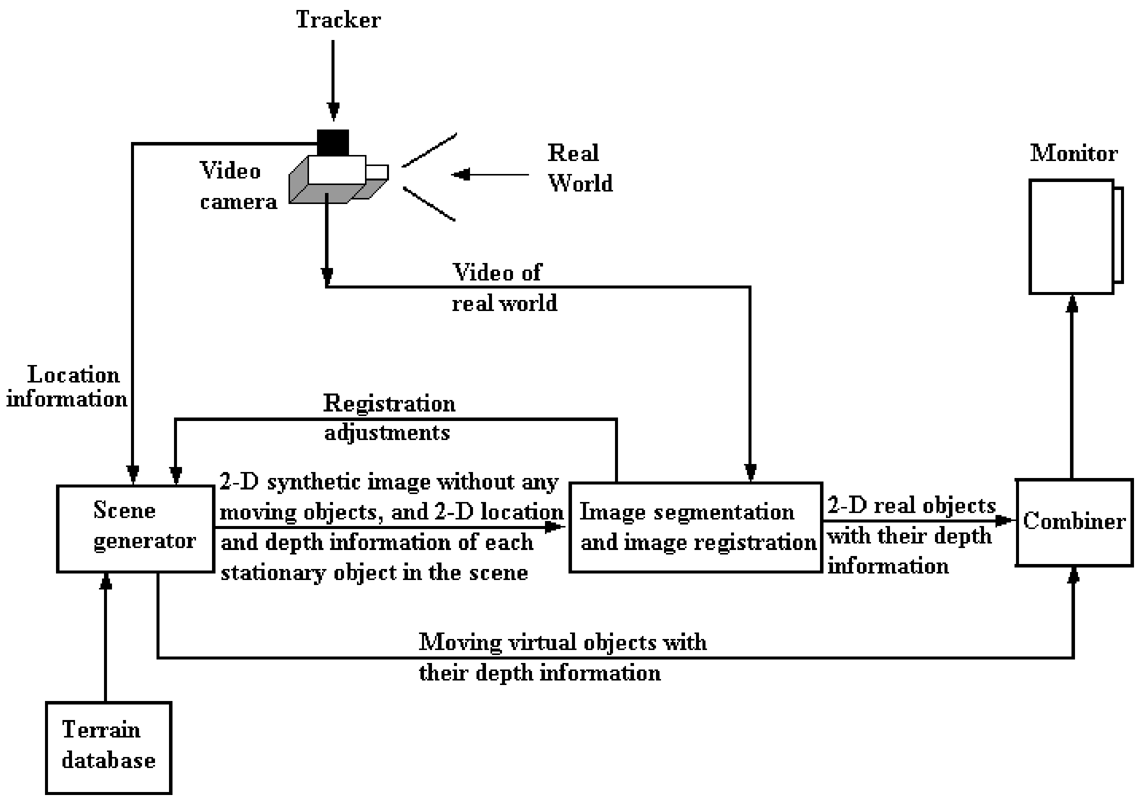

Tracker
Video camera
Real World
Monitor
Video of real world
Location information
Registration adjustments
Scene generator
2-D synthetic image without any moving objects, and 2-D location and depth information of each stationary object in the scene
Image segmentation and image registration
2-D real objects with their depth information
Combiner
Moving virtual objects with their depth information
Terrain database

Figure 3.1: System architecture.

parameters in a reasonable amount of human time.

3.2 The Augmented Reality Simulator

A schematic representation of the Augmented Reality simulator is shown in Figure 3.1. The live representation of the environment as perceived by the agent (a vehicle or a robot) is obtained from a video camera. The camera pose, zoom and speed of movement of the agent are also provided by extra sensors. This positional information is used to determine the aim-point so that we can generate a synthetic image that is roughly equivalent to the live image.

The virtual environment is represented by a visual database. Such databases include both topological and visual information. Topological data represents the geometry of the terrain, the presence and type of obstacles, types of terrain surface, etc. in a form which is suitable for use by automated tools. The visual information enables realistic rendering of the environment on a computer graphics display and usually includes surface textures and colors of the terrain surface and obstacles. We use the OpenFlight [opea] standard as an underlying visual database in our testbed.

A virtual 2D representation of the environment is generated from the OpenFlight database by rendering it with OpenGL [opeb]. Since the OpenGL rendering is often done by hardware, the computational cost for rendering the virtual environment is negligible. As part of the rendering process in OpenGL, the depth (Z-buffer) of each object in the virtual scene is determined. This depth is then used to calculate which objects are visible from the current aim-point.

A primary concern is the proper placement of the virtual objects in front of, or behind, live objects. Thus the realistic representation of the inserted objects is tied to both the appropriate occlusions and the shapes and sizes of inserted objects. A good solution to the occlusion problem requires detailed knowledge of the objects and of their location in the live scene. Since the two-dimensional live images provide no prior information about the objects in the scene, we use an image segmentation technique to segment the live image into objects.

To segment the live scene into objects we first build a look-up table for each virtual object using its color information with noise. This table is indexed by a color vector which allows us to segment the real image by applying the look-up tables to each pixel in the image. We then use a registration technique to match objects in the live image with those in the virtual scene. Depth information from the virtual scene is used to associate relative depth to each object in the live image.

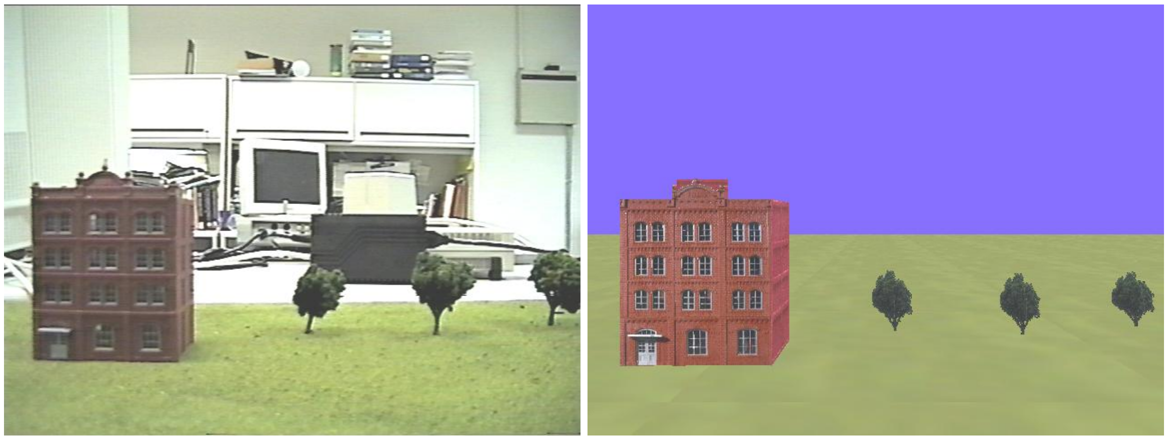


Figure 3.2: A Live scene (left) and its virtual rendering (right)

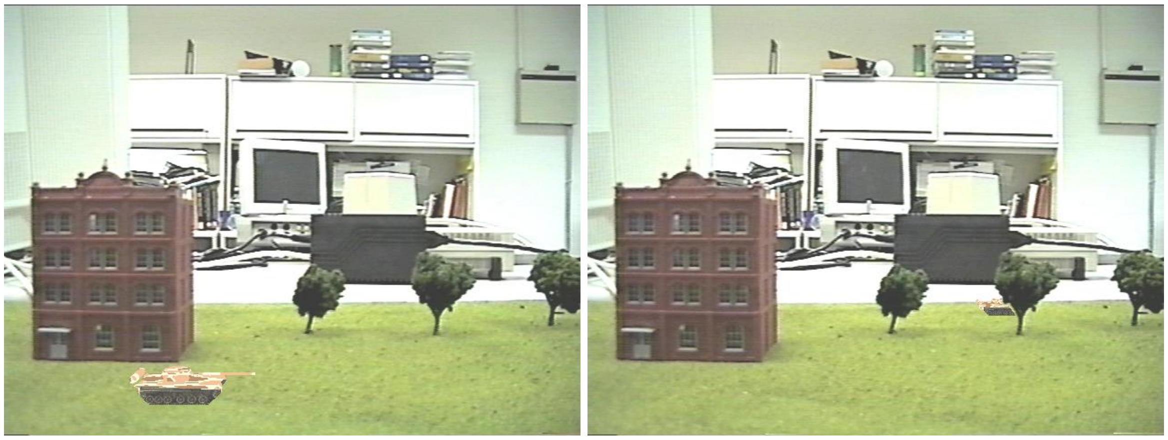


Figure 3.3: A synthetic tank included in live scenes (no occlusion on the left, occlusion with a tree on the right)

Once the depth and shape information of objects in the live scene is acquired, synthetic objects can be included with proper positioning and occlusion between real stationary objects. The block diagram in Figure 3.1 outlines the flow of data in the system. A camera provides the live terrain image. A tracker is attached to the camera to provide the location and the direction of the camera. The scene generator uses this information to generate the 2D synthetic image, and location and depth information of each stationary object in the field of view. An image segmentation algorithm uses this information to segment the live image into 2D real stationary objects.

To illustrate the approach, an example of a live image and it's corresponding virtual scene are shown in Figure 3.2. The scene is a scaled down table-top model of a terrain including a building and some trees. Note in particular the difference in shape of the trees in the synthetic and real scenes. Figure 3.3 illustrate the insertion of synthetic-target objects into the live scene - a tank in front of the building and the same tank properly occluded by the real tree in the middle.

3.3 The Agent Simulator

The Agent simulator provides an environment for developing and testing ideas in autonomous agent behavior. Currently, the environment is a 2D world which represents a terrain within which agents operate. This terrain may contain obstacles. These are represented as simple geometric shapes like circles, rectangles or general polygons.

For example, in the table-top model shown in Figures 3.2 and 3.3, the trees are represented as circles and the building as a rectangle. Figure 3.4 shows the 2D model used by the agent simulator of the table-top terrain shown earlier in figures 3.2 and 3.3.

The agent environment may also contain enemies, which are dangerous entities. Proximity of an agent to an enemy results in a positive probability of getting killed or injured by the enemy.

Naturally, the agent environment also contains the agents themselves. Each agent is aware of its position in the environment and can inquire the positions of the other agents and know how many agents are out there. The agents can also access the terrain database for checking obstacle proximity and collision by providing their position and information about their size.

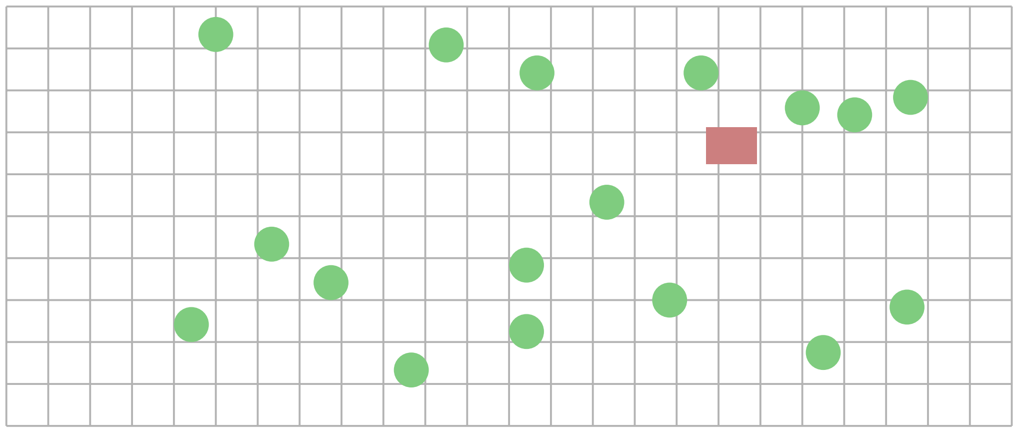


Figure 3.4: The 2D representation of the tabletop environment

3.4 The Agent Model

Our approach is based on a hierarchical modular representation of agent behavior. This method allows for de-coupling the task of group navigation into simpler self-contained sub-problems which are easier to implement in a system having computational constraints due to interaction with real-life entities.

Different decision mechanisms are used to model different aspects of the agent behavior and a higher level coordination module is combining their output. Such an architecture allows versatile agent personalities both in terms of heterogeneity (agent specialization) within a group and dynamic (i.e. mission-context sensitive) agent behavior. The hierarchical modularity of the system also facilitates the assessment of the performance of separate components and related behavior patterns on the overall success of the mission.

The following behavior modules are present in our current implementation:

- The *RNN Navigation* Module is responsible for leading a single agent from a source location to a destination location, avoiding dangerous areas on the way. The agents in this algorithm navigate using a Random Neural Network based decision mechanism which learns through Reinforcement Learning.

- The *GD Navigation* Module implements a very simple navigation approach where at each time step the agent will move in the direction which minimizes the geometric distance to destination (hence the abbreviation GD which refers to Gradient Descent).

- The *Grouping* Module is mainly responsible for keeping a group of agents together in particular formations throughout the mission. This is achieved by setting up attractive and repulsive forces between agents in order to affect their behavior.

- The *Imitation* Module is modeling the case where an inexperienced agent will try to mimic the behavior of more experienced agents in the group and thus increase its chances of success.

- The *Weapons* Module provides an agent with the capability to shoot at and possibly destroy other agents. The shooting is modeled as a probabilistic process controlled by parameters describing the firing rate, radius of effect of a weapon and probability of success.

- The *Obstacle Avoidance* Module is responsible for correcting the motion of an agent so that it does not collide with physical obstacles or other agents present in the environment.

- The *Motion Coordinator* Module is responsible for providing a weighted combination of the individual decisions of motion-related behaviors. It also enforces physical limitations on the speed and acceleration of an agent.

Besides the core behaviors described above, the simulator provides an API for adding other behaviors. This API allows behavior modules written in C++ to be added to a simulation *dynamically* at run-time. The simulator can also be extended through an embedded high-level scripting language called Lua [IFF96]. Lua is simple yet powerful dynamically-typed language designed specifically for extending applications. It is particularly well suited to augment C and C++ programs by providing a rapid-prototyping capability. We use it extensively in most of the examples described in Chapter 4 for implementing mission-specific behavior.

3.5 Agent Behavior Modules

The individual decisions proposed by the above behavior modules need to be combined in some manner in order to provide a final course of action for an agent. Our implementation allows for connecting an arbitrary number of behaviors in many possible configurations. However, in most cases we are primarily interested in groups of agents which can move within an environment and can exhibit adversarial action towards each other.

We consider the motion and adversarial parts of agent behavior as orthogonal. This properly implies that we can generate these parts independently. The part of agent action related to motion is generated by the Motion Coordinator module. Broadly speaking, this module will fuse the individual decisions of the motion-related behaviors and apply some corrections (simple obstacle avoidance and speed limitations). The adversarial part of agent action, on the other hand, is decided by the Weapons Module alone without any direct involvement of other behaviors. This limitation,

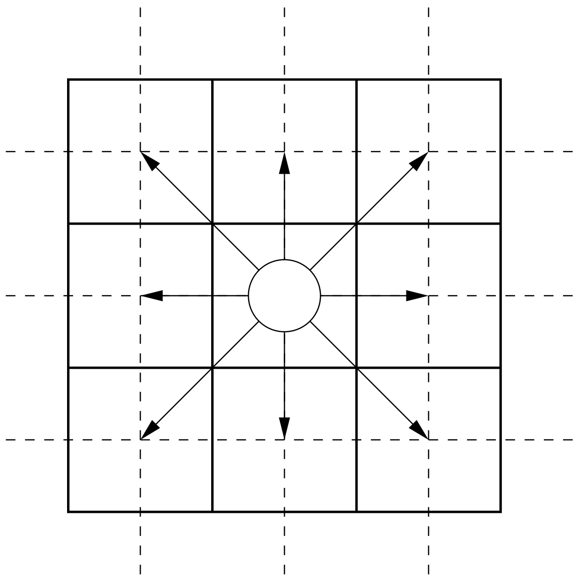


Figure 3.5: Representation of an 8-neighborhood

however, can easily be overcome by providing extra mission-specific behaviors and functionality through the simulator API.

The following sections describe the operation of the behavior modules in detail.

3.5.1 RNN Navigation

The RNN navigation module is one possible way of implementing a goal-based navigation for an agent. This module incorporates a quantized grid representation of the agent world. Such a quantization allows a relatively simple and efficient storage of knowledge and experience relevant to the navigation task in the otherwise continuous environment. Obviously, there is a requirement that the scale of quantization should be chosen carefully so that the relevant information is relatively uniform within each grid cell.

Each cell in the grid represents a position and an *agent action* is defined as the decision to move from a grid cell to one of the eight neighboring cells (Figure 3.5). A succession of such actions will result of a completion of a mission. The agents can access terrain-specific information about features and obstacles of natural (trees, etc.), and artificial origin (buildings, roads, etc.) and also sense the presence of other (possibly hostile) agents. The interaction between an enemy (a hostile agent) and an agent is modeled by an associated risk. This risk is expressed as a probability that the agent will be shot while being at a position which is within the firing range of an enemy. The goal of the agent is to minimize a function G (which in this case is the estimated time of a safe transit to the destination). We use G to define the Reinforcement Learning Reward function such that $R \propto 1/G$.

Successive measured values of R are denoted by R_l, $l = 1, 2, \ldots$. These values are used to keep track of a smoothed reward T_l, defined as

$$T_l = bT_{l-1} + (1-b)R_l, \quad 0 < b < 1$$

where b is a number close to 1. An agent has a so-called *cognitive map* which is a collection of latest and smoothed rewards for each decision taken at each visited grid cell.

The decision-making element of an RNN Navigation Module is a fully inter-connected RNN network consisting of 8 neurons (each representing a possible decision). The training is performed by reinforcing the weights of each neuron, depending on the difference between the latest and smoothed rewards; positive difference indicates improvement and negative difference indicates deterioration. A detailed description of the network and the learning process are given in Appendix A, Appendix B and in [GSX01].

By using previously acquired information and current sensory input, an agent can start with

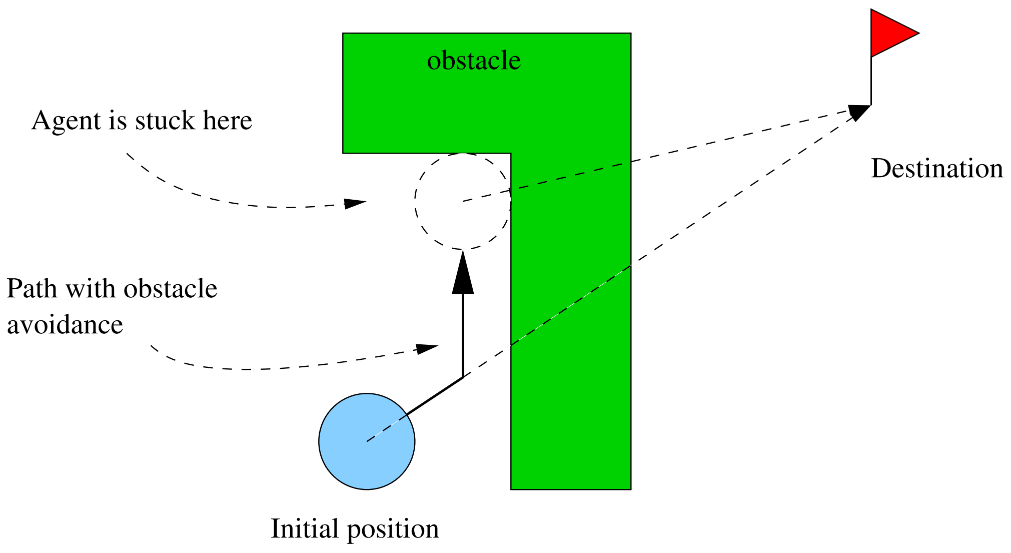


Figure 3.6: Illustration of the local minimum problem that GD navigation can encounter when operating in an environment with obstacles.

fairly reasonable estimates of the rewards and skip an otherwise prohibitively-long learning session and focus on refining its knowledge and adapting to the dynamic changes in the environment.

3.5.2 GD Navigation

GD Navigation is a simpler and cheaper alternative to RNN Navigation. While RNN has a memory in the form of a cognitive map, a GD Navigation Module has no state except the current position of the agent and location of the destination. Whenever a GD navigation module is asked to make a decision, the answer provided will be to go directly towards the final goal position.

Being a completely reactive approach to navigation makes the GD module susceptible to getting stuck in a local minima when navigation interferes with obstacle avoidance. This problem is illustrated in figure 3.6 - the agent's intention to move towards the destination is counteracted by the requirement that it should not physically penetrate obstacles. There is no simple way to avoid this problem within the framework of reactive navigation alone, mainly due to the lack of a memory component where previous behavior can be encoded. As a result, the action suggested by the GD navigation is dependent only on the metric of the Cartesian space of the environment and not on the features of the particular terrain in which the mission is being executed.

The reason why we consider this mode of navigation at all, is for comparison with more complex navigation algorithms like RNN navigation. GD navigation is the simplest possible mode of navigation in terms of requirements on computational resources. It also illustrates very well the main drawbacks of reactive navigation when applied to a complex environment.

3.5.3 Grouping

The operation of the grouping module is based on the idea of social potential fields. In our treatment, we restrict the form of the force between agents i and j to:

$$\vec{V}_{i,j} = \left(-\frac{a}{r^\alpha} + \frac{b}{r^\beta} \right) \hat{r}$$

where a, b, α, β are dynamic parameters of the force, r is the distance between the agents, $\hat{r}$ is a unit vector pointing from i to j, and the vector $\vec{V}_{i,j}$ represents the effect of the position of agent j on the decision of agent i. The terms $(-a/r^\alpha)$ and (b/r^β) represent the *attractive* and *repulsive* components of the SPF force. A graphical illustration of these components and the effect of their combination is shown in Figure 3.7.

When there is a stable equilibrium point, an entity experiencing such a force will try to stay at a distance R_0 from the force source, where

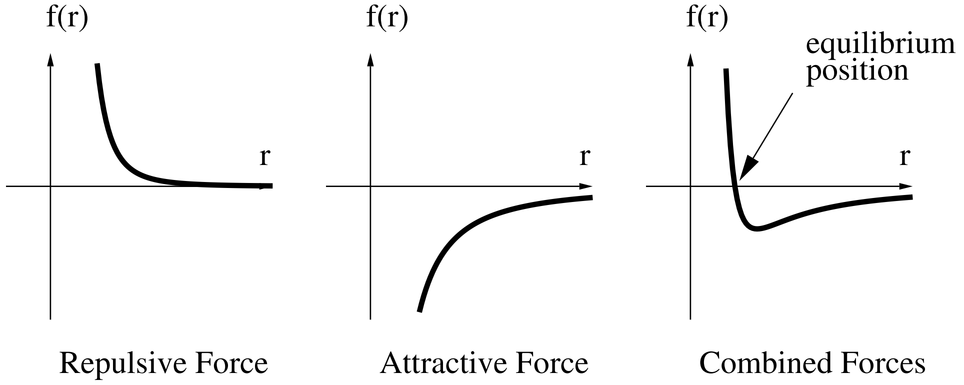


Figure 3.7: Graphs of the components of an SPF force: repulsive(left), attractive(middle) and a combination of both (right).

$$R_0 = \sqrt[\beta-\alpha]{\frac{b}{a}}$$

The total force acting on agent i can be calculated as the vector summation of individual forces due to all other agents:

$$\vec{V}_{grp_i} = \sum_j \vec{V}_{i,j}$$

A wide range of emergent behaviors can be achieved by varying the parameters of forces and the force configurations between agents or groups. For example, a group can be “encouraged” to stay together in a spatially localized formation by setting up a collection of two-way forces (with both attractive and repulsive components) between each member of the group. Another example is a configuration of one-way repulsive forces between a group and its adversaries in order to encourage it to avoid enemy contact. SPF forces can also be used to model the effects of the *collision avoidance* and *flock centering* rules in the *Boids* architecture described by Reynolds [Rey87].

3.5.4 Imitation

The imitation module generates a decision which is a weighted sum of the navigational decisions of some of the members of the agent group:

$$\vec{V}_{imt_i} = \sum_{j \in S} w_j * \vec{V}_{nav_j}$$

The weight distribution (w_j) can be dynamic, in order to reflect the group members which are currently observable or known to be experienced, for example. The purpose of incorporating imitation in the behavior of an agent is to try to take advantage of the experience of other agents without going through the trouble of actually acquiring it - that is, it opens the possibility of "harnessing" the experience of other agents in a very efficient manner.

The *velocity matching* flock behavior described in the work of Reynolds [Rey87] which he defines as the "attempt to match velocity with nearby flocks" is functionally very similar to the imitation module.

3.5.5 Weapons

There are two parameters which control how an agent is involved in an adversarial action against other agents. These are the *firing rate* (λ) and *firing range* (r). When an agent detects one or more enemies within its firing range, it will engage its weapon and target the closest one. Weapons operate in distinct shots which are modeled as a Poisson arrival process with parameter λ. The probability p of the success of a shot depends linearly on the distance d between the agents and is computed by the following formula:

$$p(d) = \begin{cases} 1 - d/r & , d < r \\ 0 & , d \geq r \end{cases}$$

According to this formula, p approaches 1 when an enemy is very close with respect to the firing range, and decreases to zero as distance to the enemy increases towards the firing range limit. The outcome of each shot is determined as a hit or miss depending on the on the respective value of p.

3.5.6 Obstacle Avoidance

The purpose of this module is to provide a rudimentary method for enforcing the physical constraint that an agent cannot penetrate obstacles in the environment or other agents. It is a purely reactive approach which does not include any kind of planning.

The module operates by taking as an input a motion decision in the form of a vector. This decision is checked for *probable* collisions by examining the immediate environment of the agent (terrain obstacles or other agents). A collision is considered *probable* if the agent will collide with an obstacle when moving in the suggested direction for the next t seconds (a configurable look-ahead parameter).

If a probable collision is detected, the module will search for smallest possible angular deviation (to the left or to the right of suggested motion vector) which will clear the collision. If such a deviation exists, the module will output a corrected vector as the new decision, otherwise it will decrease the speed of the agent until the collision is no longer probable.

3.5.7 Motion Coordination

This module combines the decisions of all behaviors which affect the motion of an agent[1]. It will also apply corrections associated with obstacle avoidance and physical limitations on speed and acceleration. First, a combined motion decision is generated as a weighted vector sum of the individual agent decisions:

$$\vec{V}_{\text{motion}} = \sum_{i \in N} k_i * \vec{V}_i$$

In this formula, $\vec{V}_{\text{motion}}$ represents the combined decision, N is the set of available motion-related decisions, $\vec{V}_i$'s are the individual decision vectors and k_i's are their respective weights.

The combined decision $\vec{V}_{\text{motion}}$ is then processed by the Obstacle Avoidance module for corrections. The final operation consists of clamping the vector to ensure that the maximum speed and acceleration of the particular agent are not exceeded.

3.6 Performance Metrics

The behaviors described above are quite simple and although it may be relatively easy to analyze them individually, it is difficult to judge empirically what their effect will be on the overall performance of agents. In order to evaluate and compare the performance of different behavior combinations, we have devised a number of simple metrics which quantify certain aspects of agent behavior. These metrics are:

[1]Such behaviors produce decisions which are 2D vectors.

1. Group Tension
2. Group Radius
3. Travel Distance
4. Travel Energy
5. Group Speed
6. Number of Survivors

Of the above, *Group Tension, Group Radius* and *Number of Survivors* are metrics which only make sense when defined for groups of agents. On the other hand, the *Travel Distance*, *Travel Energy* and *Speed* metrics can be defined for both individual agents and groups. The group versions of these metrics will usually be defined as the average of the individual metrics of the agents that belong to the respective group.

The following subsections contain a detailed explanation of which aspects of behavior these metrics are good indicators for, and how they are measured.

3.6.1 Group Tension

During execution of a mission each agent will experience a number of SPF forces. The total effective force, which is the vector summation of the individual forces exerted by each agent, will indicate the direction in which the Grouping module of this particular agent *wants* to go. A reasonable way to measure how well a particular agent formation reflects the intention of the force configuration is to average the magnitudes of the effective total forces exerted on each agent. We call this value the *group tension*. The tension T_g of an agent group g with n agents is calculated as:

$$T_g = \frac{1}{n}\sum_{i=1}^{n}\left|\vec{F_i}\right|$$

where $\vec{F_i}$ is the total force experienced by agent i of group g.

A small value for the group tension should indicate that the group is well-formed, while a bigger value should indicate internal stress within the agent group related to bad spatial formation. In particular, the value will become very large when agents come very close to each other. The tension will also increase if the group is slowed down when some of the agents have trouble following the rest due to encountering obstacles or congestion. However, the group tension is not a good indicator for "pathologically" malformed groups where some agents have separated from the main formation or groups that have completely broken into spatially-distinct units. The reason is that the magnitude of SPF forces decays with distance and agents which are too far from the group formation will contribute very little to the group tension.

3.6.2 Group Radius

The group radius is defined as the average distance of agents to the group center, where the group center is defined as the arithmetic mean of the positions of all agent belonging to a group. In a sense, the group center is the center of mass of the agent group. If group g has n agents with positions $\vec{p_i}, i = 1..n$, then the group center $\vec{c_g}$ is calculated as

$$\vec{c_g} = \frac{1}{n} \sum_{i=1}^{n} \vec{p_i}$$

and the group radius R_g is calculated as

$$\vec{R_g} = \frac{1}{n} \sum_{i=1}^{n} |\vec{p_i} - \vec{c_g}|$$

An increase in group radius can be used to detect cases when some agents have lost spatial contact with their respective group. A decrease from a normal value, on the other hand, will indicate that the group has shrunk due to congestion.

3.6.3 Travel Distance

The travel distance metric is defined as the average distance traveled by all agents in a group as a function of time. Essentially, this is the average of the values shown in the odometers of each agents, assuming they have ones. We calculate the travel distance for an agent i with position $\vec{p_i}(t)$ as

$$D_i(t) = \sum_{s=0}^{t} |\vec{p_i}(s) - \vec{p_i}(s-1)|$$

and the travel distance for a group g with n agents is calculated as

$$D_g(t) = \frac{1}{n} \sum_{i=1}^{n} D_i(t)$$

There are two aspects of performance related to this metric. First, during the execution of the mission, a higher value of travel distance will indicate that a group is moving faster - which may or may not be a good thing depending on the specific mission objective. The second aspect comes out when the mission is finished and all agents stop moving. A lower stationary value for travel distance would then indicate that the agents finished the mission while traveling less distance, which usually would be considered to be a good thing.

3.6.4 Travel Energy

Defining how much energy is spent by agents during a mission is a very difficult task. The problem is that different types of agents (e.g. a human, an autonomous robotic platform or an armored vehicle) will have very different energy spending profiles. The nature of the terrain will also have a great impact on how much energy is consumed.

Since our agent model is working at a higher-level of abstraction, we are considering an energy model inspired by basic physics. Any object with a positive mass m that is moving with a speed v with respect to an inertial reference frame has a kinetic energy K defined as:

$$K = \frac{1}{2}mv^2$$

The amount of energy δK needed to increase the speed from v_1 to v_2 is equal to the difference between the respective kinetic energies:

$$\delta K = K_2 - K_1 = \frac{1}{2}m(v_2^2 - v_1^2)$$

At this point, we should note that for vehicles in general, it is much easier to reduce kinetic energy than to increase it. This is due to the fact that converting kinetic energy into heat and dissipating it does not usually require any technology more complex than break pads and an actuator pedal (this is how breaking in cars or bicycles works, for example). Increasing kinetic energy, on the other hand, is a process that generally requires some kind of an engine and a fuel to operate it. Therefore, we define the *travel energy* metric for an individual agent as the sum of the positive kinetic energy increments[2] as a function of time in the following way:

$$E(t) = \sum_{i=0}^{t} \delta K_i^+$$

where

$$\delta K_i^+ = \begin{cases} 0 & \text{, if} \quad (v_i < v_{i-1}) \quad \text{or} \quad (i \leq 0) \\ (v_i^2 - v_{i-1}^2) & \text{, otherwise} \end{cases}$$

The travel energy of a group g with n members is calculated as the average travel energy of its members:

$$E_g(t) = \frac{1}{n} \sum_{i=1}^{n} E_i(t)$$

The definition of the travel energy implies that an agent with a smoother motion, where the speed does not change very frequently (an everyday example would be highway traffic) will *spend* less energy than an agent which changes speed frequently (like the motion pattern in a congested city traffic). This may not be a very realistic model to apply to humans traveling on foot, for example, since people cannot operate continuously for prolonged periods and may need frequent breaks

[2]we drop the constant $\frac{1}{2}m$ for simplicity

when traveling in a tough terrain, especially when carrying heavy support equipment. Another example where this analogy may not be applicable is heavy armored vehicles like tanks, where considerable amount of energy is spent just to sustain a certain speed. We should therefore point that the *travel energy* as defined here is a good indicator of how jittery a motion is, not how much real energy is spent by an agent. We call it *energy* because of its relation to kinetic energy in physics.

3.6.5 Group Speed

The group speed is defined as the average speed of all agents in a group. For an agent i with position $\vec{p}_i(t)$, the speed can be calculated as

$$V_i(t) = \sum_{s=1}^{t} |\vec{p}_i(t) - \vec{p}_i(t-1)|$$

and the group speed for a group g with n agents is calculated as

$$V_g(t) = \frac{1}{n}\sum_{i=1}^{n} V_i(t)$$

This metric may seem like an unnecessary redundancy, since from physics we know that speed is defined as the time derivative of travel distance, which we already measure. We provide it separately because, as it will become obvious in the following discussion of experimental results, most of the time it is difficult to resolve differences in group speeds by looking at travel distance alone.

3.6.6 Number of Survivors

The last metric that we will consider is the number of survivors in a group, $S_g(t)$. As the name implies, it is defined as the number of agents in group g that are alive at time t. Agents will be eliminated when adversarial groups with weapons capability are within firing range of each other and shoot at each other successfully. Obviously, it is good for a group to have a higher value for this metric.

CHAPTER 4

EXPERIMENTS

4.1 Introduction

This chapter describes in detail three experiments that we have performed in the Autonomous Agents simulator. The first experiment is designed to provide insight into the quantitative effect of individual behaviors on the observed action of agents and their overall success. The purpose of the second experiment is to illustrate how the simulator testbed can be used as an aid in a decision making process by providing quantitative information on how certain agent parameters affect the expected outcome of a hypothetical military scenario. This scenario is also used in the third experiment, where we illustrate the advantages of an extensible and flexible agent architecture by integrating mission-specific behaviors into the core of the agent model. Specifically, we enhance some of the agents by providing them with means to disperse a chemical and examine its effects on enemy units and on the overall success of the mission.

The last section in this chapter presents some observations related to our experience with multi-agent simulations. First, we discuss why it is difficult to provide statistically significant answers to questions of interest without comprehensive sampling of the problem space by running many simulations. We also discuss the importance of emergence of unintended behavior and the difficulties faced by behavior-based agents when navigating in an environment with complex obstacles. These two problems, in particular, form the basis of our further research and are addressed in detail in Chapters 5 and 6.

4.2 Experiment 1: Comparative Analysis of Behavior Performance

The purpose of this experiment is to show how the combination of simple behaviors can improve the overall success of groups of agents operating in an adversarial situation. This is accomplished by analyzing the performance metrics of groups of agents under different modes of operation.

4.2.1 Description

The terrain of this experiment is a square area with a size of 2000 by 2000 units. This terrain contains obstacles in the form of trees and buildings. There are 6000 trees and 100 buildings. The trees are represented as circles with with radii picked randomly from the range between 2 and 3 units. The buildings are represented as rectangular obstacles with sizes changing between 5 and 8 units. The positions of both trees and obstacles are randomly generated from a uniform distribution.

There are two groups of agents designated as *blue team* and *red team*. The blue team consists of 8 agents (1 leader and 7 group members). The mission of this group is to move from one location in the terrain to another. This group may or may not have weapons.

The red team consists of 1, 3 or 5 agents. The primary mission of this group is to intercept the blue team and destroy it. It is therefore always equipped with weapons.

A smaller version of this type of terrain is presented in Figure 4.1 (300 by 300 units, 200 trees and 10 buildings). For the purpose of showing the size of the groups and the agents that constitute them, the blue team is also depicted in the process of moving towards a destination.

The configuration of the blue team is as follows:

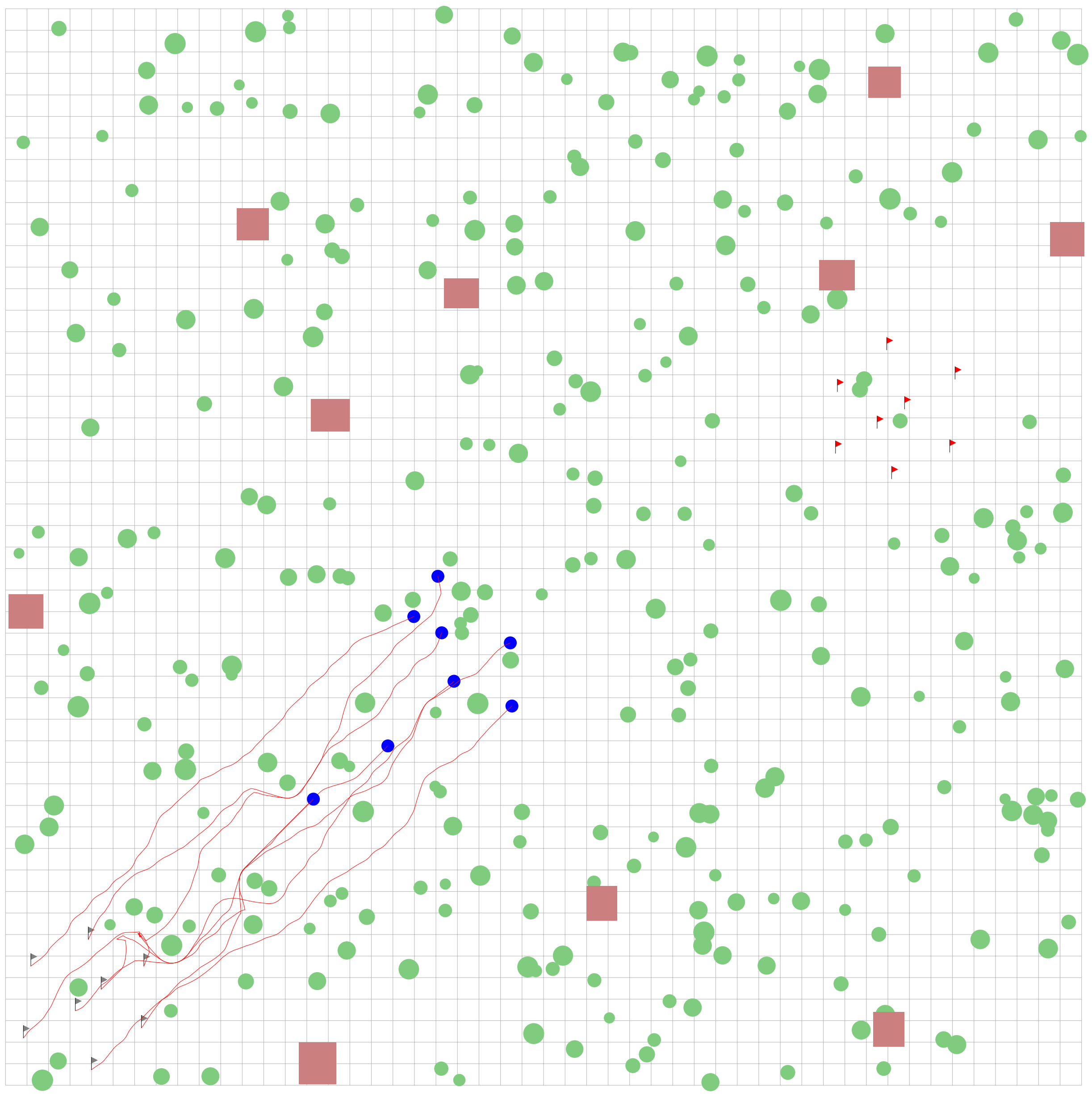


Figure 4.1: Example of a terrain similar to the one used in experiment 1. The agents of the blue team are in the process of moving from their initial positions (lower-left corner) to their final destination (top-right corner).

- Each agent moves by RNN and GD Navigation (only one of these is active during the course of a single simulation run).

- There are two types of SPF forces:

  1. A two-way force between group members (excluding the leader). The force parameters are $a = 1, \alpha = 1.6, b = 16, \beta = 3.6$.

  2. A one-way force from the leader to the group members. The force parameters are $a = 1, \alpha = 1.6, b = 4, \beta = 3.6$.

  These parameters have the effect of keeping an inter-group distance of approximately 4 units, and distance between group members and the leader of approximately 2 units - in this way, the group is surrounding the leader. The decision on the leader is not directly dependent on the other agents.

- Team members are configured to imitate their leader. The imitation module of the leader is always disabled.

- Weapon parameters: firing rate is 1 shot per second, weapon range is 5 units.

- Agent parameters: The radius of the agents is 0.3 units, maximum speed is 3 units per second, maximum acceleration is 0.3 units per second square.

The configuration of the red team is as follows:

- There is a one-way force from each member of the blue team to each member of the red team. The force parameters are $a = 1, \alpha = 1.6, b = 0, \beta = 0$. As a result of this force, the red agents are attracted to all members of the blue team.

- Weapon parameters are the same as the blue team.
- Agent parameters are the same as the blue team.

A number of simulations sets are executed with the following options being explored:

- The blue teams uses either RNN Navigation or GD Navigation.
- The weapons of the blue team are either enabled or disabled.
- The red team has 1, 3 or 5 agents.

The total number of simulation sets is 12 ($2 \times 2 \times 3$). Each simulation set collects group measurements for the blue team averaged over 1000 different missions, where a mission is defined as a set of initial and final positions for the blue team. The missions are selected randomly, subject to the constraint that the distance between initial and final positions is between 49% and 51% of the diagonal size of the terrain. The initial position of the red team is fixed at the middle of the terrain. We measure the performance metrics for four different behavior *modes*. These are navigation-only, navigation and grouping, navigation and imitation, and all three enabled. The difference between the behavior modes will provide insight into the effects of each behavior on the overall performance.

The average duration of a mission was experimentally found to be approximately 2000 time steps. Since not all missions finish at the same time, we ran the simulations for 3000 time steps (corresponding to 100 seconds of mission time).

4.2.2 Results

Due to the large amount of information gathered during the simulation runs, we provide the detailed experimental results in Appendix C.

Table 4.1 provides a qualitative comparison of the performance. Each column represents a particular configuration set and the order (1 to 12) is the same as the order of the figures in Appendix C. Each row in the table represents a particular metric type (one of the following: *group radius*, *group tension*, *travel distance*, *travel energy*, *number of survivors*, *group speed*). The table consists of 4 major parts (marked as NGI, N, NG and NI). These refer to the behaviors enabled during a mission run:

NGI - Navigation, Grouping and Imitation enabled.

N - Only Navigation enabled.

NG - Only Navigation and Grouping enabled.

NI - Only Navigation and Imitation enabled.

A measurement of a metric in a each behavior mode is marked with a black square (■), gray square (▩) or a white square (□) if it is considered to be within the set of *best*, *average* or *worst* measurements respectively, where the comparison is performed over the four behavior modes (NGI, N, NG and NI). Empty space is used to indicate the cases in which the difference between all modes is insignificant or inconclusive. The difference is insignificant for the *number of survivors* and the *travel distance* metrics in columns 7–12. Also, the measurements of the *travel distance* metric in columns 1–6 are considered inconclusive because although some of the behavior

Table 4.1: Characterization of the performance metrics for the 12 simulation runs. Black, gray and white boxes indicate that the metric was among the best, average or worst, respectively. Empty space indicates that difference between measurements was considered insignificant.

		1	2	3	4	5	6	7	8	9	10	11	12
NGI	group radius	■	■	■	■	■	■	■	■	■	■	■	■
	group tension	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
	travel distance												
	travel energy	■	▨	■	▨	■	▨	■	▨	■	▨	■	▨
	survivors	▨	▨	▨	▨	▨	▨						
	group speed	□	□	□	□	□	□	■	■	■	▨	□	□
N	group radius	□	□	□	□	□	□	□	□	□	□	□	□
	group tension	□	□	□	□	□	□	□	□	□	□	□	□
	travel distance												
	travel energy	□	■	□	■	□	■	▨	■	▨	■	▨	■
	survivors	■	■	■	■	■	■						
	group speed	■	■	■	■	■	■	■	■	■	■	■	■
NG	group radius	■	■	■	■	■	■	■	■	■	■	■	■
	group tension	■	■	■	■	■	■	■	■	■	■	■	■
	travel distance												
	travel energy	□	□	□	□	□	□	□	□	□	□	□	□
	survivors	□	□	□	□	□	□						
	group speed	□	□	□	□	□	□	□	□	□	□	□	□
NI	group radius	□	□	□	□	□	□	□	□	□	□	□	□
	group tension	□	□	□	□	□	□	□	□	□	□	□	□
	travel distance												
	travel energy	■	■	▨	■	▨	■	■	■	■	■	■	■
	survivors	■	■	■	■	■	■						
	group speed	■	■	■	■	■	■	■	■	■	■	■	■

modes (i.e. NG and NGI) appear to perform better, this is due to a coupling between number of survivors and travel distance and not because they actually are better (the experiments are designed so that the geometric distance between starting and destination positions is fixed at approximately 1400 units).

Based on the experimental results, we make the following observations:

- It is evident that while single behaviors tend to improve certain aspects of agent performance, they do so at the expense of adversely affecting other metrics. However, the performance improvement of the NGI mode (where all three behaviors are combined) shows clearly that even very simplistic combination techniques like ours are able to provide a more comprehensive advantage than the constituent behaviors can do by themselves. This observation is consistent with the general belief of researchers in behavior-based control that “the whole is bigger than the sum of its constituents”.

- GD navigation seems to perform much better than RNN navigation for easy terrains. This is most apparent in the measurements the travel energy metric, which show approximately 5-fold increase from GD to RNN. We believe this is due to grid artifacts introduced by the fact that RNN navigation operates over a discrete quantization of a continuous terrain. This issue is discussed further in section 4.5.2.

4.3 Experiment 2: Measuring Effects of Agent Speed

The purpose of this experiment is to illustrate how our system can be used to explore the effects of parameters on the success of a mission. Specifically, we are looking at how the speed of a support team will affect the survival chance of agents in a particular military scenario involving collaborating and competing groups.

4.3.1 Description

The terrain used in this scenario is shown in Figure 4.2. There are three teams designated as *Red*, *Blue* and *Green* team. The Blue and Green teams are friendly forces while the Red team is their enemy. The mission objectives of each team are as follows:

- The goal of the Blue team (located in the lower-center part of the terrain) is to advance in a group formation to their final location (the building close to the the top-right corner of the terrain). It will also try to avoid contact with the Red team.
- The goal of the Red team (located close to the center of the terrain) is to intercept and destroy the Blue team.
- The goal of the Green team (located in the lower-left part of the terrain) is to support the Blue team by following and engaging the Red team.

The terrain size is 300 by 300 units and contains 200 trees and 1 building. Each team consists of 5 agents. The configuration of the blue team is as follows:

- Each agent moves to its destination by GD navigation.
- Agents are encouraged to stay together through two-way SPF forces. The force parameters are $a = 1, \alpha = 1.6, b = 16, \beta = 3.6$.

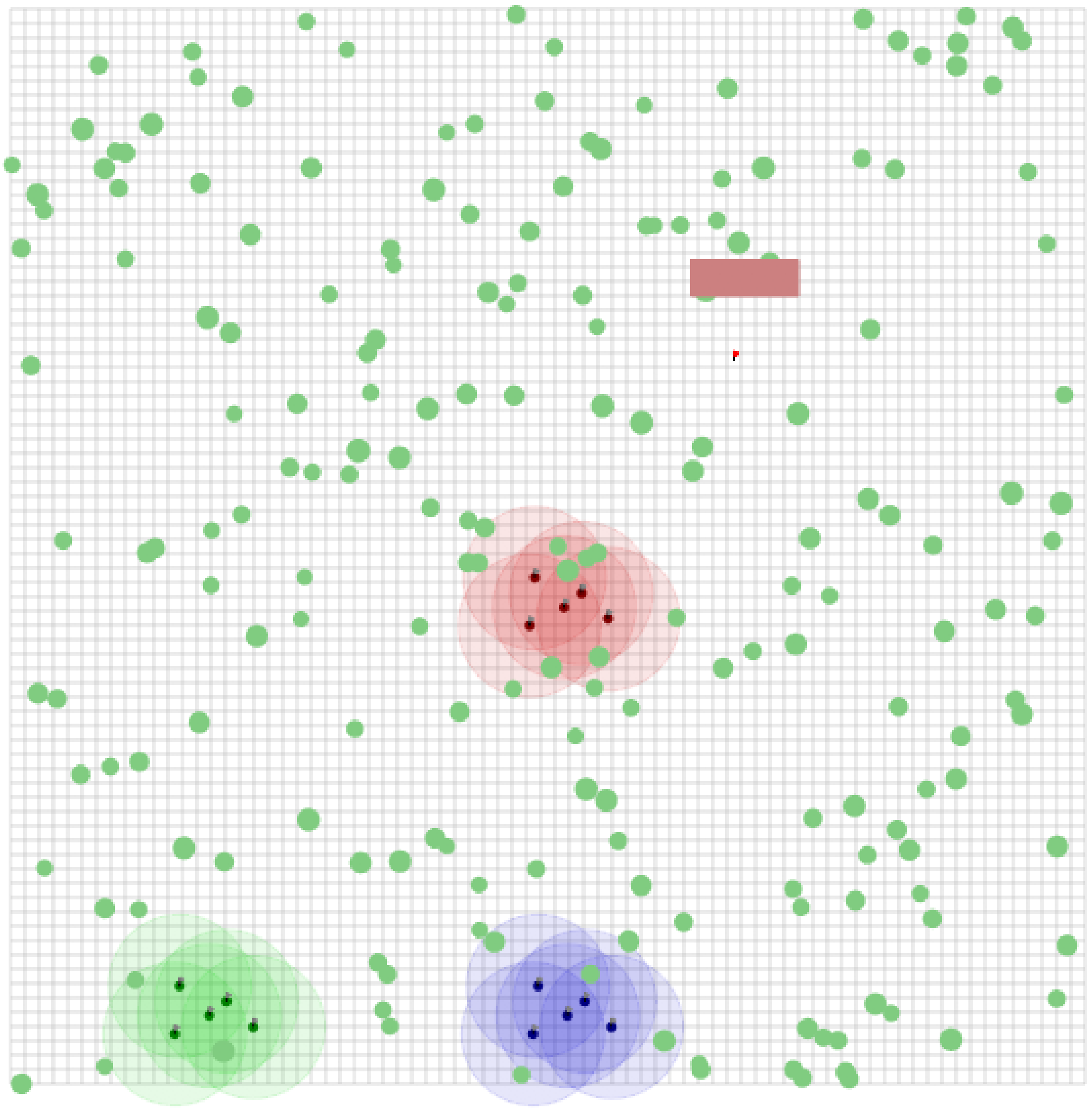


Figure 4.2: Terrain and mission configuration used in Experiment 2

- There is a one-way repulsive force from the red team with parameters $a = 0, \alpha = 1.6, b = 4000, \beta = 0.36$.

The configuration of the green team is as follows:

- Agents are encouraged to stay together through two-way SPF forces. The force parameters are same as the blue team.
- There is a one-way attractive force towards the red team with parameters $a = 1, \alpha = 1.6, b = 0, \beta = 3.6$.

The configuration of the red team is as follows:

- Agents are encouraged to stay together through two-way SPF forces. The force parameters are same as the blue and green team.
- There is a one-way attractive force towards the blue team with parameters $a = 1, \alpha = 1.6, b = 0, \beta = 3.6$.
- There is a one-way repulsive force from the green team with parameters $a = 0, \alpha = 1.6, b = 4, \beta = 3.6$.

Other agent parameters (size, speed and acceleration limitations, weapons range and firing rates) are identical to the parameters of the previous experiment (described in Section 4.2).

We run five sets of experiments where the speed of the green team varies between 2 and 4 in increments of 0.5 (the speed of the other teams is fixed at 3). For each set we measure the expected number of survivors in each team over 500 missions by varying the random number generators

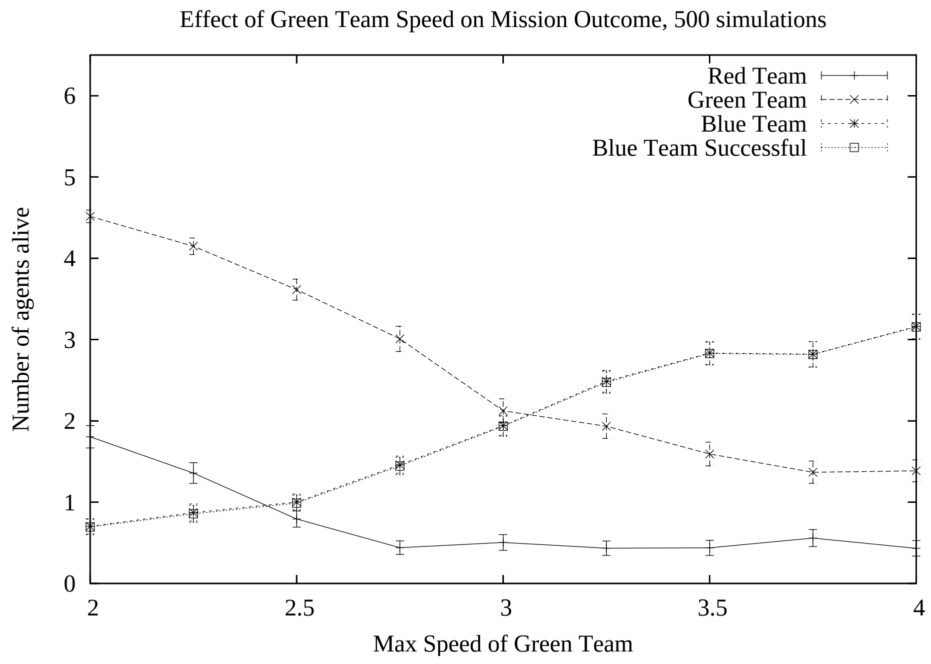


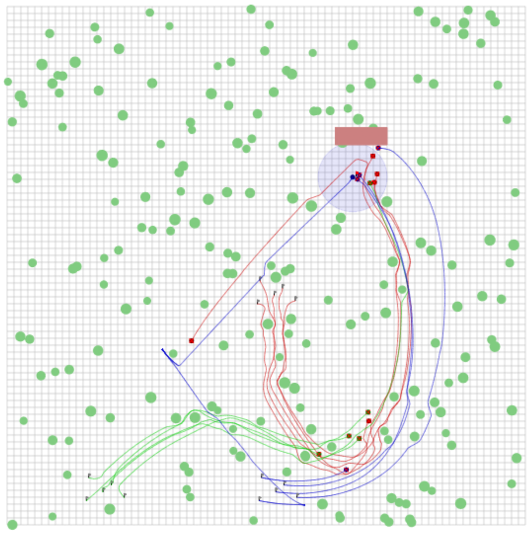
Figure 4.3: Effects of speed of green team on survival rates and mission success.

associated with weapons fire for each mission. We also measure the number of successful survivors for the blue team, where success is defined for an agent as being alive and being located at the destination. Simulation time is limited to 2000 steps.

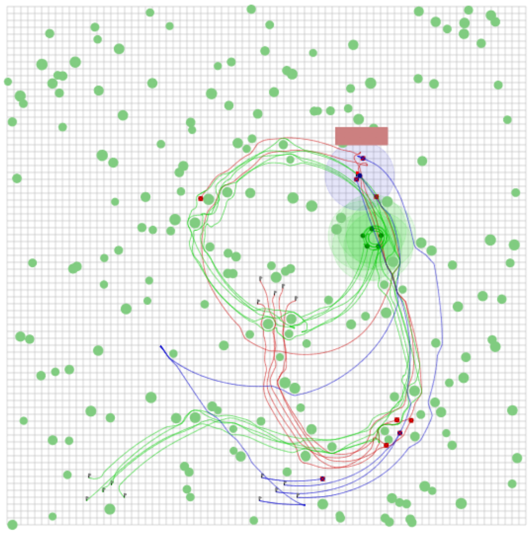
4.3.2 Results

The change in the expected number of survivors as a function of simulation time is shown in Figure 4.5. The overall effect of the speed of the green team on mission success is shown in Figure 4.3. It is evident from these results that an increase of the speed of the green team above 3.5 does not provide a statistically significant increase in success.

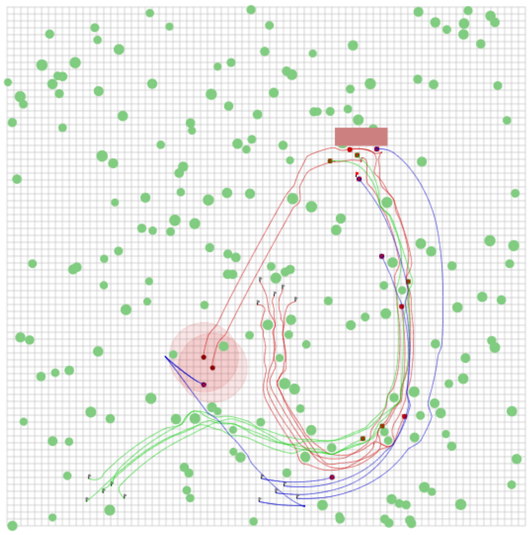
Another interesting observation is that two simulation runs which are identical except for the randomness associated with weapons fire, can result in significantly different outcomes. This is due



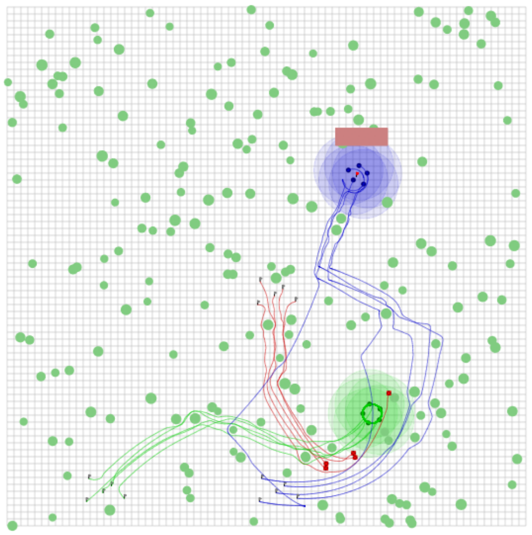
(a) a single blue agent survives



(b) 1 blue and 5 green agents survive

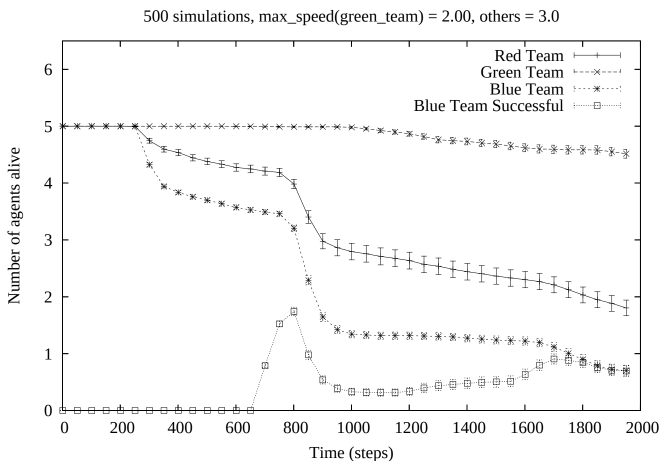


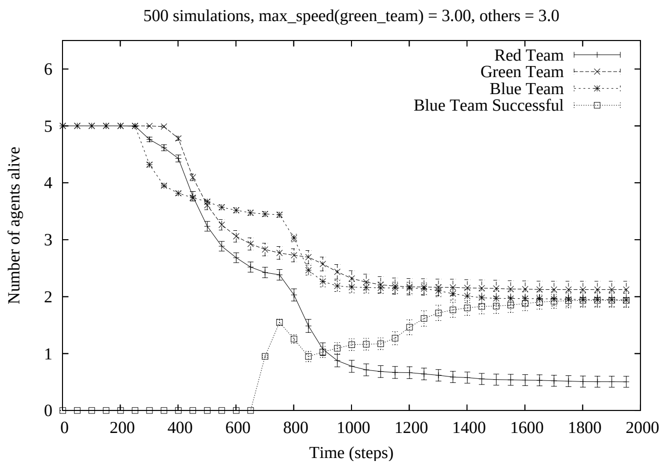
(c) 2 red agents survive



(d) 5 blue and 5 green agents survive

Figure 4.4: Examples of different mission outcomes due to randomness.





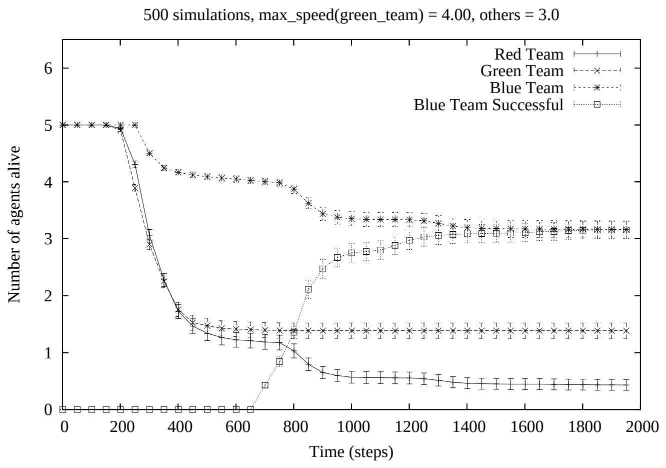


Figure 4.5: Evolution of the number of survivors as a function of time for different speeds of the green team.

to the fact that elimination of agents through shooting introduces changes which have significant long-term effect on the outcome of a mission. This implies that obtaining statistically significant results requires a comprehensive sampling of the problem space (e.g. by integrating measurements over many simulation runs).

4.4 Experiment 3: Measuring Effects of Chemical Dispersal

This experiment is almost identical to the previous one with the exception of the addition of an extra capability to one of the members of the blue team. This agent is able to release a chemical which slows down the red team but has no effect on the blue and green teams.

4.4.1 Description

The terrain, the agent teams and their behavior configurations are exact duplicates of the experimental setup described in Section 4.3 and will not be described again. The novelty in this experiment is the introduction of mission-specific behavior (adversarial action through chemical dispersal) and the purpose is to illustrate the extensibility of our simulator platform. All of this extra functionality is not a core part of the simulator but a set of extensions which are dynamically attached to the system as needed.

The operational semantics of the process of chemical release can be described as follows:

1. One member of the blue team has cans full with a chemical which when released will slow down the red team but will not affect the blue or green teams.

2. The agent is instructed to activate and drop the cans when it detects red agent in its proximity

(in this experiment, the detection radius is 40 units).

3. When activated, the cans will release their contents at a constant rate until depleted (in this experiment, the chemical is depleted in 300 time steps). Due to its nature, the chemical will spread over an area and eventually dissipate.

4. The effect of the chemical is to reduce the maximum speed of the red team. The maximum reduction corresponds to half their original speed and happens when the concentration is at maximum at a location.

In the simulator, this process is represented through a stochastic population model based on a queuing network. The terrain is quantized to a 75×75 grid with each cell representing a discrete location. The concentration of chemical at each location is represented as the number of customers waiting in an associated M/M/1 queue. The rate of removal of chemical from each location (due to diffusion into neighboring locations or dissipation) is represented as the service rate μ_i of the associated queue. The process of diffusion and dissipation are governed by the probabilities $p_{i,j}$ that a customer will move to queue j after being serviced at queue i. The customers may also leave the queuing system with probability $(1 - \sum_j p_{i,j})$. Customers enter a queue i either due to diffusion from a neighboring cell or due to the presence of an active undepleted can at that particular location. The effect of active cans is modeled as an external arrival of customers with rate λ_i.

The following values for these parameters were used in the experiment:

1. $\lambda_i = 100$, for all i.

2. $\mu_i = 100$, for all i.

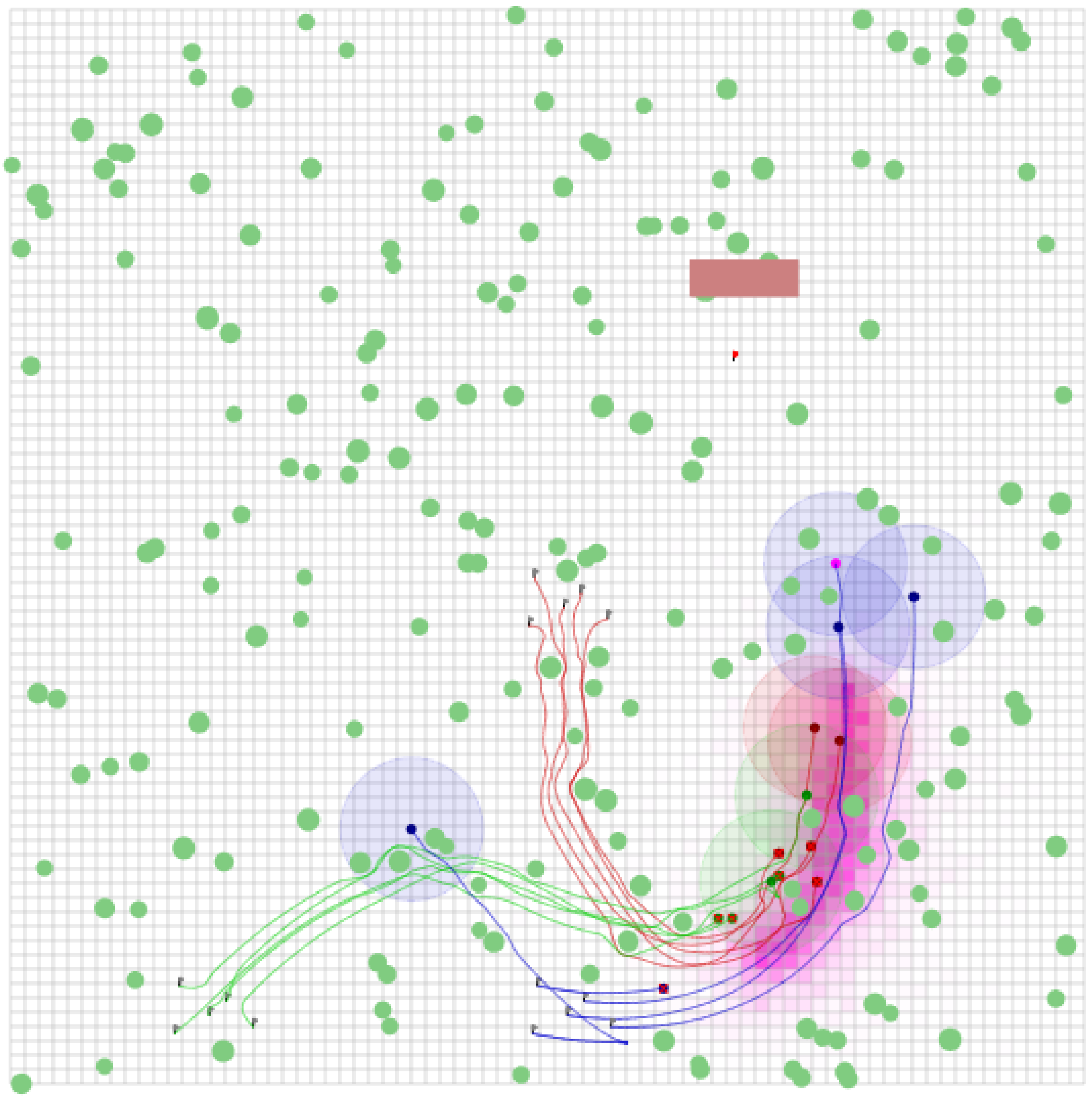


Figure 4.6: A screenshot of the simulator running a mission with chemical dispersal.

3. For each location i, $p_{i,j} = 0.1125$, for all j in the 8-neighborhood of i. As a result, customers diffuse with probability $\sum_j p_{i,j} = 0.9$.

4. At each location i, the probability of dissipation is 0.1 $(1 - \sum_j p_{j,j})$.

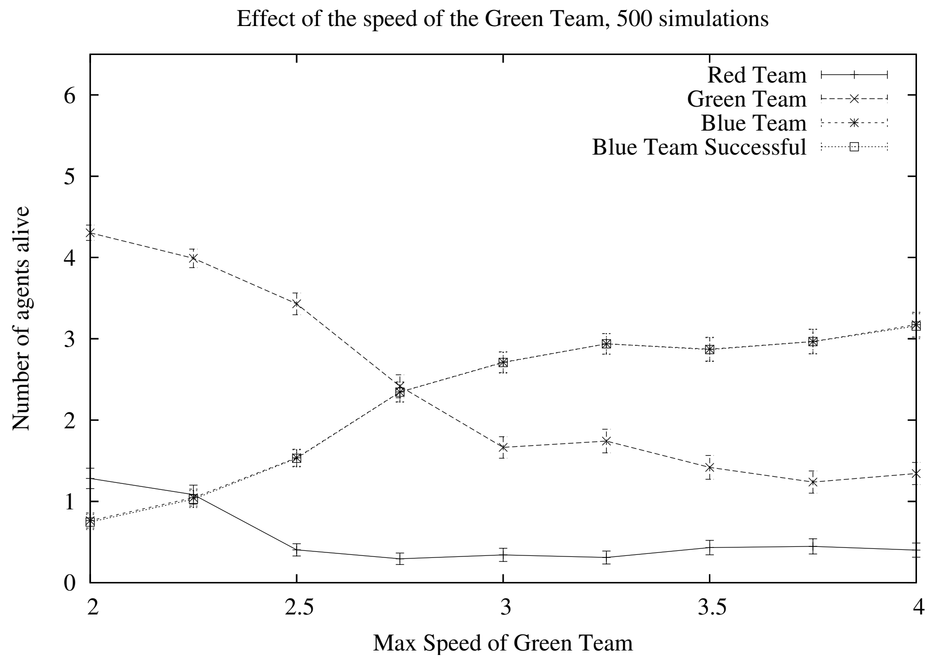


Figure 4.7: Effects of the speed of the green team on the survival rates and mission success when the chemical is used.

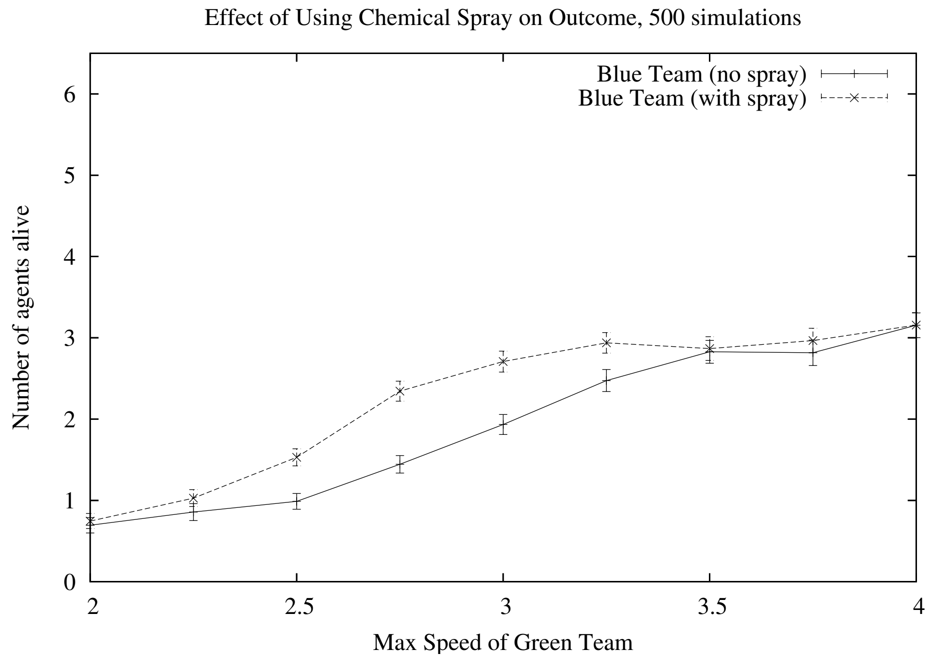


Figure 4.8: Improvement of the survival of the blue team due to chemical spray as a function of the speed of the green team.

4.4.2 Results

Figure 4.6 shows a screenshot of an ongoing mission in the simulator. The agent of the blue team with with the chemical dispersal capability is marked in pink. The concentration of chemical in the environment is also represented as various shades of pink. The effect of the speed of the green team on the survival rates of teams is shown in Figure 4.7. Figure 4.8 shows a comparison of the survival rate of the blue team with and without chemical dispersal.

It is evident from these results that using a chemical to slow down the red team is statistically effective only when the difference between the speeds of the enemy (red team) and the support group (green team) is less than 0.5 units. Using a chemical outside this range does not provide any advantage to the blue team. However, a closer inspection of Figures 4.3 and 4.7 reveals that improvement to the survival rate of the blue team when using chemicals is achieved at the expense of a reduction of the survival rate of the green team.

4.5 Discussion of Observations

This section includes detailed discussions of a number of important observations related to the interpretation of the experimental results presented in this chapter. First, we will provide a qualitative description of how non-determinism and chaotic behavior affect the process of providing answers of statistical significance. Then, we will identify two research problems which we feel need to be addressed further. These are the phenomenon of emergent behavior and the problem of reactive navigation in an environment with complex mobility constraints. These discussions will lay the groundwork for the following two chapters, where we provide solutions to these problems.

4.5.1 Obtaining Estimates of Expected Agent Behavior

One of the reasons for performing simulations is to evaluate possible approaches for solving a problem by choosing a plan of action which is optimal in some sense. For example, in our case we may be interested in how well a group of agents will perform given a particular set of behaviors or what types of behaviors will produce a desired outcome. There are two main reasons why this is a difficult problem:

1. The agent model includes a stochastic component (i.e. adversarial action) which makes the overall system non-deterministic.
2. Even if we ignore the stochastic component, some parts of the deterministic components of behavior (i.e. SPF) are inherently chaotic.

Here is a description of what we mean by *determinism* and *chaos* in the above statement:

- Determinism implies that given a set of initial conditions, we can in principle compute the exact state of the system at any given time to a reasonable degree of numerical accuracy.
- Chaos is a concept from the field of Dynamic Systems. In Physics, *Dynamics* is the study of systems which change with respect to some parameter (usually time). One of the important contributions of research in non-linear dynamic systems is the concept of *chaos*. A deterministic system is considered *chaotic* if it behaves in an aperiodic manner such that very small changes in initial conditions can generate very large differences in the outcome as time advances [Str94]. The importance of this property is that it renders prediction of long-term behavior in chaotic systems impossible.

Our claim that the deterministic part of the agent model is chaotic is based on the observation that Social Potential Fields can be used to represent the gravitational interaction between a number of objects and it is a well-known fact that the 3-body problem exhibits chaotic behavior [BM93].

The non-deterministic and chaotic nature of agents makes it very difficult to give statistically robust answers to questions about the expected behavior of the system by running only a few simulations. The above problems are not specific to our particular agent model or implementation - on the contrary, we are describing them in the context of a relatively simple behavior model in order to emphasize the fact that such problems are an integral part of most multi-agent systems.

This is, in fact, one of the main reasons for our interest in simple reactive methods of agent control. We believe that computationally efficient methods similar to our approach are more likely to offer a framework for exploring alternative courses of action in a timely manner within the constraints of currently available technology. This is true of course, provided that the methods in question are able to approximate the interaction within the system that is being simulated to a satisfactory degree of accuracy.

4.5.2 Navigation and Complex Environments

Our main reason for experimenting with a RNN-based reinforcement learning algorithm for navigation was the hope that such an approach would enable agents to acquire information and experience about the environment in which they operate and use it to their advantage. Specifically, we wanted the agents to discover routes which would allow them to accomplish their missions better. However, our simulation results showed that, while computationally efficient, our approach was hardly effective, mainly because the of the prohibitively long learning times. The value of this

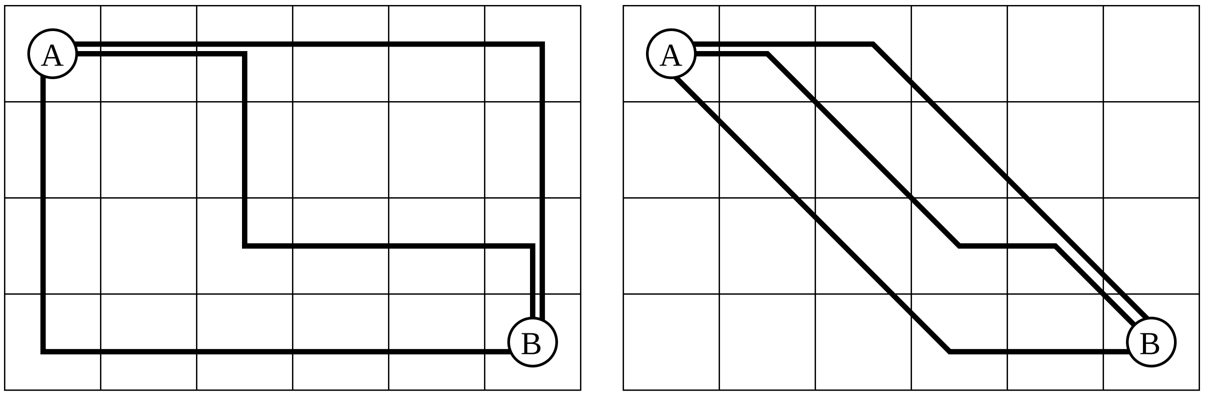


Figure 4.9: Examples of multiple shortest paths between two positions (A and B) under 4-neighborhood (shown on the left) and 8-neighborhood quantization (shown on the right).

methods is diminished further by the fact that the learned experience is both terrain and mission-specific and cannot be reused directly in other circumstances. While it may be possible to mitigate this problem by using more sophisticated methods of learning, such methods tend to come with an excessive increase in complexity, both computational and otherwise.

Table 4.2: Distance metrics of continuous 2D euclidean space and its quantizations.

Name	**Functional Form**
Euclidean Distance	$D_e = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
Manhattan Distance	$D_m = \lvert x_2 - x_1 \rvert + \lvert y_2 - y_1 \rvert$
Chebyshev Distance	$D_c = \max(\lvert x_2 - x_1 \rvert, \lvert y_2 - y_1 \rvert)$

Another major problem with the RL RNN navigation is related to the fact that this algorithm operates over a quantized representation of the environment. It is a well known fact that the performance of such algorithms tends to suffer from *grid artifacts*. Sometimes it is possible to mitigate such effects (Yap, for example, uses a tiled hexagonal quantization as an alternative [Yap02]) but removing them completely is sometimes impossible. This problem is illustrated in Figure 4.9 for square grids with 4 and 8 neighbors. In both cases, the quantization changes the underlying metric of the space. As a result, any algorithm trying to solve a problem defined in terms of the metric of the space ends up solving the wrong problem. Table 4.5.2 show the 2D versions of the distance metrics for continuous Euclidean space and its 4-neighborhood and 8-neighborhood quantizations. The latter two are better known as *Manhattan distance (or L_1 distance)* and *Chebyshev distance*.

Our efforts to address this problem led to the development of a novel navigation algorithm. The approach we take has very low runtime computational requirements and is not affected by grid artifacts because it operates in a continuous domain. This algorithm is described in Chapter 5.

4.5.3 Dealing with Emergent Behavior

Reactive behavior-based methods of control in multi-agent systems can be a very efficient tool for simulation of large systems. Unfortunately, these methods have the undesirable property of being unpredictable and cannot generally provide any guarantee of success. This is mainly due to the fact that the actions of agents in such systems are built from a number of components which are not capable of solving the *big* problem, but rather try to "encourage" the expression of certain behaviors with the hope that a combination of such behaviors will contribute towards achieving a common goal.

Let us look for example at Figure 4.4 where four different outcomes of the same mission are shown. Examination of the tracks of the blue team reveals that one of the blue agents consistently fails to stay within the spatial formation of the team. This a clear violation of the intention imposed by the SPF forces to keep the agents together. However, such situations may be of particular interest as a way of modeling the effect of human factors. The renegade agent in our example has a much higher chance of survival and thus a higher chance of completing the mission (such a case is shown in Figure 4.4 (a)).

In military conflicts, unexpected situations are the norm rather than the exception. One of the best expressions of this statement is the quote *"No battle plan survives contact with the enemy"*, attributed to Field Marshal Helmuth, Graf von Moltke[1]. Therefore, we believe that working towards building a capability to detect and react to surprises is a much better approach than trying to avoid them through premature planning. To this end, we have developed a method of tracking groups based on spatial and behavioral similarities. As it will be seen from the experimental results, this algorithm is particularly well suited for application in the context of our research. Chapter 6 is dedicated to this subject.

[1]Also known as *Helmuth Karl Bernhard von Moltke*, a German Field Marshal, Chief of staff of the Prussian army for thirty years (October 26, 1800 - April 24, 1891).

CHAPTER 5

NAVIGATION IN URBAN ENVIRONMENTS

5.1 Introduction

The major change of context of modern military conflicts away from rural areas and into the confines of densely populated urban environments has posed unique challenges for tactical planning of military operations. These challenges require a new set of technologies and methods to enable efficient and effective operation within the constraints of urban environments as well as exploit as much as possible the specific features of the city terrain.

One of the new requirements is the ability to operate in an environment containing a large number of civilians where it may be very difficult to selectively target an enemy force. Failing to preserve civilian life can have dire consequences on the overall success of a military operation. Another operational constraint is related to the requirement to preserve civilian infrastructure. Among other things, this constraint implies limitations on the mobility of forces.

In the previous chapter we explored the possibility of modeling the behavior of teams of autonomous agents with common goals through purely reactive methods like social potential fields and gradient descent navigation. The most attractive feature of such control methods is their tendency to scale extremely well when the number of mobile agents becomes very large. Another possible advantage is the ability to build complex scenarios by defining a small number of broadly-defined goals. Instead of being explicitly described, the behavior of individual agents in such systems becomes an emergent property.

Unfortunately, these methods also come with a major disadvantage. One of the most important problems in the context of multi-agent simulations is the severe decrease of success of reactive agents when the environments in which they operate impose non-trivial mobility constraints.

In the rest of this chapter we will introduce a novel algorithm for efficient navigation in environments containing complex obstacles. We describe our approach within the context of urban environments, which is a particularly relevant application area.

The traditional approach to navigation in such domains has been based largely on path planning algorithms. Such approaches tend to be computationally expensive and therefore do not scale well when the number of agents in a simulation increase. Scalability is also affected by dynamic changes in the environment, since these may invalidate original plans and force the agents to compute new ones.

Our approach, on the other hand, is largely reactive and scales very well with the increase of the agent population. Moreover, since our algorithm is based on the idea of transforming the real space into a virtual obstacle-free space, it can serve as an adapting tool for applying reactive or greedy algorithms into domains where they traditionally do not perform very well.

It is interesting to note that while space deformations have been used extensively in many areas – for example animation and visual effects [KCP92, YHM04, FM98] or computer-aided modeling [TM91, GM05] – we are not aware of any research into space deforming transformations as a tool for terrain navigation.

The failure of reactive behavior control techniques to perform well in complex environments can be characterized as a local minima problem. Figure 3.6 illustrates such a situation in the context of agent navigation. The name *local minima* refers to a commonly occurring situation in the field of Mathematics when searching for parameters which globally maximize or minimize the value of a function. Simplistic approaches to search can easily fail by getting trapped at a local extremum point.

In the context of reactive navigation of groups of agents, a local minimum situation is not always considered detrimental, especially when it is due to internal agent dynamics rather than as a result of interaction with the environment. For example, we use social potential fields in order to encourage group formation by providing agents with a behavior component which minimize the forces acting on them. In this context, achieving the spatial formation corresponding to a global minimum is not essential - what is important is the "intent" to minimize.

In navigation, the most common example of an undesirable local minimum situation occurs when a collision avoidance scheme adversely affects the motion towards an intended goal and as a result the agent either becomes stationary or is stuck on a localized cyclic path, thus failing to achieve its objective. These situations are the primary target domain of our navigation algorithm.

5.2 Terrain Transformation

We propose a method of avoiding obstacles that is based on the idea of finding a transformation between the real navigation space and a virtual obstacle-free space and applying the classical agent control methods within this new space. In this section, we will describe how such transformations can be computed. As a first step, we will try to illustrate the idea in a more informal but accessible way. This introduction is followed by a formal mathematical description of the transformation.

5.2.1 Informal Description

The following description presents the main idea of transforming space in what we hope is a relatively easy to understand fashion:

1. We assume that the terrain is represented as a 2D birds-eye view of the environment. All obstacles have a continuous boundary which is a closed contour. The interior of the obstacles is not accessible to agents. The space outside the contours is free space.

2. We start the process of transforming the real space into a virtual obstacle-free space by continuously shrinking the obstacle contours inwards until all the obstacles collapse into point singularities. The free space is *attached* to the contours and will *stretch* in the process of obstacle collapse.

3. When all the obstacles have collapsed, the new stretched free space would span the whole real terrain and any point in this virtual space will be accessible from any other point trivially (by navigating along a straight line, for example).

4. It is very important that the transform conserves the local continuity of the space in the process – this property guarantees that any continuous paths (straight lines, for example) in the *stretched* space are continuous curves in the real space, and therefore, valid navigation paths.

5.2.2 Formal Description

It is not difficult to realize that there are infinitely many ways of performing the above transformation. What we will describe below is one approach which happens to be particularly easy to compute and is also well suited for obstacle avoidance.

Figure 5.1 shows an example of the types of terrains that we are interested in. The obstacles are represented as polygons with their interior painted in gray, their boundary outlined and a cross mark at what can be considered the obstacle center. The obstacle center is the point into which the obstacles will “collapse”. We do not provide an exact definition of the center because, technically any point inside an obstacle is a valid choice and will work with our algorithm. In practice, a good candidate is the center of mass of an obstacle[1] or the location of the maximum of the fundamental vibration mode of a surface stretched over the shape of the obstacle.

The particular collection of obstacles and their respective centers shown in Figure 5.1, for example, was randomly drawn by hand in order to provide a reasonable set of non-convex shapes that are commonly found in urban areas. This data set will be used throughout the rest of this chapter to describe the algorithm and provide experimental results.

5.2.3 *The Terrain Potential*

The definition of the terrain transformation depends on a real-valued function over the terrain that we call the *terrain potential*. This potential can be computed in the following way:

1. We quantize the terrain into a grid. At the end of the computation each grid cell will have an associated potential (a real number between -1 and 1, inclusive).

2. Cells which contain obstacle centers have their potential fixed at -1.

3. Cells which include an obstacle boundary have their potential fixed at 0.

[1]Except in cases when the center of mass happens to be outside the boundaries of an obstacle

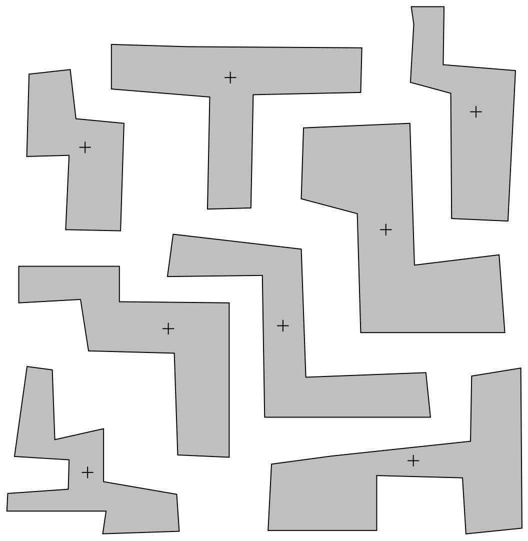


Figure 5.1: Terrain with a number of non-convex obstacles

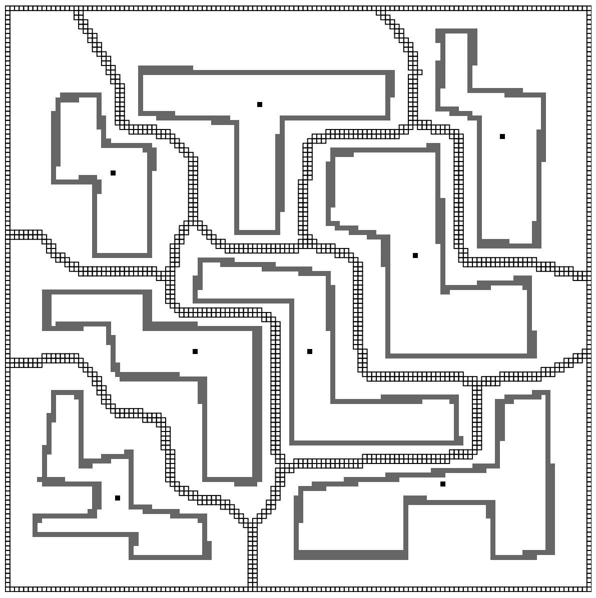


Figure 5.2: Boundary conditions (grid cells with fixed potentials)

4. Cells that are completely inside or outside obstacles have their potential initialized to -0.5 and +0.5, respectively (this step is not necessary, but will help in faster convergence).

5. For each free cell (i.e. on the outside of obstacles), we compute the distance to the nearest obstacle. Cells which have neighbors closer to obstacles other than their own, have their potential fixed at +1. The same is done for cells at the boundary of the grid.

6. We run an iterative process where at each step the potential of each cell (excluding cells with already fixed potentials) is replaced by the average potential of its 4 neighbors. The iteration continues until the potential converges.

Figure 5.2 shows cells with fixed potential for our example terrain (quantized at 128×128). Cells with potentials fixed at -1, 0 and +1 are represented as black filled squares, gray squares and white outlined squares, respectively.

The formal mathematical explanation of the above process is that we are using Jacobi iteration to obtain a numerical solution of the Laplace equation subject to Dirichlet boundary conditions.

Physicists will easily recognize that the above procedure is equivalent to finding the electric potential ϕ of a system of electrical conductors (hence the name *terrain potential*), where cells at the center of obstacles represent a conductor with a potential fixed at $\phi = -1$, cells at obstacle boundaries correspond to a conductor with a potential fixed at $\phi = 0$, cells at an equal distance between distinct obstacles represent conductors with $\phi = +1$ and all other cells represent free space (vacuum).

Since the algorithm described below operates in continuous space, we use a quadratic filter [BMD02] to reconstruct a continuous terrain potential from the quantized numeric solution.

Figure 5.3 shows the shape of the equipotential lines of ϕ for the test environment (quantization at 256×256, 2000 iterations). Note that the equipotential lines approximate the obstacle boundary as ϕ approaches 0. As the potential decreases towards -1, the shape of the lines converges smoothly towards a circle. Also, as the potential increases towards +1, the shape of the lines becomes more convex.

5.2.4 Transforming the Terrain

Once the terrain potential has been obtained, we can find a transformation which will convert the real terrain into an obstacle-free virtual terrain. Note that the potential of the exterior of obstacles (i.e. accessible space) is limited to the range $(0; 1]$ and that of the interior of obstacles (inaccessible space) is in the range $[0; -1]$. The virtual position $\vec{p'}$ of any point $\vec{p}$ in the real terrain can be

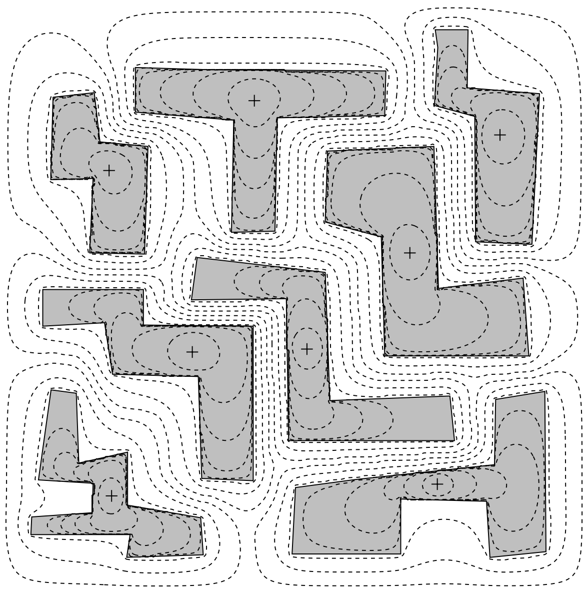


Figure 5.3: Equipotential profile

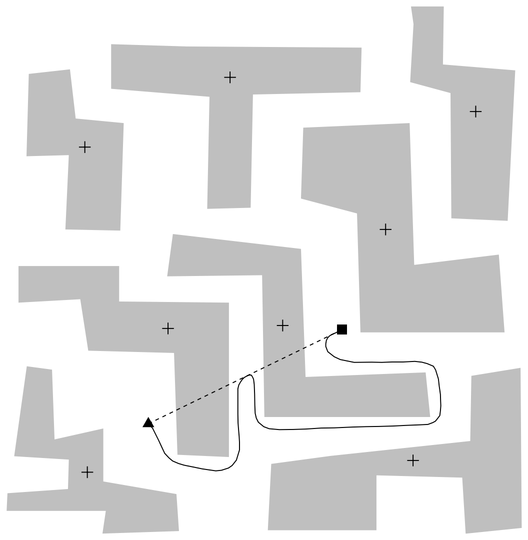


Figure 5.4: Shape of a virtual straight line in real space

found[2] by moving down the gradient of the terrain potential (starting at $\vec{p}$) until a point $\vec{p'}$ with potential $\phi(\vec{p'}) = \max\{f(\phi(\vec{p})), -1\})$ is reached, where $f : \mathbf{R} \rightarrow \mathbf{R}$ is any function which is continuous and monotonically increasing in the range $[0;1]$ and for which $f(0) = -1, f(1) = 1$ and $\forall x \in [-1;0], f(x) \leq -1$. Since there are infinitely many functions with this property, we are actually defining a class of transformations isomorphic to the class of functions f with the above properties. For simplicity, from now on the term *transformation* will refer to any of one these.

Notice that when such a transformation is applied to all points in an obstacle interior it will collapse the obstacle to a point singularity coinciding with its respective center. Also, when applied to all other points, the transformation will expand free space to cover all of the terrain except a finite number of point singularities at the obstacle centers. Moreover, the transformation is reversible when the range is limited to free space.

[2]Technically, this is true when there are no saddle points in the terrain potential. An example of when this can happen is an obstacle which is almost completely surrounded by another. However, this is not a problem for our algorithm since we do not really compute the transformation itself.

The importance of these properties lies in the fact that instead of solving the problem of collision-free path between two points in the real terrain, we can instead solve the corresponding problem in the virtual space and apply the inverse transform to obtain a valid solution in the real space. The reason why this approach is attractive is that, due to the lack of obstacles in the virtual terrain[3], any continuous curve that connects two points in the virtual space corresponds to a valid path between their respective real counterparts. The simplest case of such a curve is a straight line. Figure 5.4 shows a sketch of how a straight line in virtual space might look when mapped back onto the real terrain. The starting point and the goal are represented with a triangle and a square, respectively. The dotted curve represents the shape of the straight line in real space.

At this point, we should note that while the computation of the terrain potential is a very straightforward and efficient procedure, obtaining a terrain transform from this potential is not. This is mainly due to the fact that gradient descent towards a point singularity over a uniform approximation of such potentials is numerically a very unstable.

Fortunately, computing such a transformation is not necessary. Nevertheless, it is presented here for the sake of providing a better understanding of why we use such an approach. For example, the path shown in Figure 5.4 can be generated by smoothly switching between (1) going towards a destination, (2) going around an obstacle (following the equipotential lines), and (3) back to going towards the destination after approaching the exit point at the other end of the obstacle. All the necessary information for this algorithm can be extracted from the local terrain potential. Note also that while not providing optimal paths, at least a collision-free path to destination is guaranteed (if one exists, of course). In the next section we will introduce a heuristic approach which can do

[3]Except for point singularities which can be avoided trivially.

better in terms of discovered path length at the expense of success ratio.

5.3 A Heuristic Navigation Algorithm for Urban Environments

In this section we will describe a heuristic approach to agent navigation based on the idea of terrain potential. The algorithm described below is just a proof of concept (mostly for the purpose of illustrating the idea and initial experimentation) and we are confident that a more careful and systematic construction will yield better results. Experimental results show that while not being able to guarantee a successful discovery of a path, this algorithm appears to performs quite well on our test terrain data.

The algorithm is based on performing a number of computations at each step. The outcome is a direction of motion and once the decision is applied the agent moves to the new position and the process is repeated until the destination is reached. These computations require the availability of the local terrain potential and the center of the closest obstacle. To be precise, the decision process depends on the following information:

- terrain potential at current position
- center of closest obstacle
- internal state variables (destination and an internal variable s)

Note that the local nature of the required information allows it to be stored in completely distributed data structures. Table 5.1 describes our notation in detail.

Here is an algorithmic description of the steps necessary for obtaining a motion decision:

Table 5.1: Explanation of the notation used for describing the heuristic navigation algorithm

ϕ	terrain potential at the current location
$\hat{n}$	normalized gradient vector of ϕ (i.e. $\hat{n} = \frac{\vec{n}}{\vert\vec{n}\vert}$, where $\vec{n} = \vec{\nabla}\phi$
d	distance from current position to destination
d_{min}	threshold dependent on terrain scale
$\vec{d}$	a vector from current position to destination
$\hat{d}$	unit vector pointing towards destination
$\hat{c}$	unit vector pointing towards current obstacle center
s	current avoidance direction (left, right or none)
$\hat{s}$	a unit vector in the current avoidance direction (always perpendicular to $\hat{n}$)
$\vec{a}$	a vector representing decision outcome (i.e. direction of motion)

1. if a change in $\vec{c}$ is detected at the last step (*i.e. moved from one obstacle's field of influence into another*) then

   $s \leftarrow$ 'none'

2. if $\hat{n} \cdot \hat{d} \geq 0$ then *(agent is climbing the terrain potential – i.e. moving away from obstacle)*

   (a) if $\phi < 0.9$ then

      i. if $s =$ none then pick s such that $\hat{d} \cdot \hat{s} > 0$

      ii. if $\hat{d} \cdot \hat{s} < -0.2$ then $\vec{a} \leftarrow \hat{s}$, otherwise $\vec{a} \leftarrow \hat{d}$

   (b) otherwise $\vec{a} \leftarrow \hat{d}$

3. otherwise *(agent is descending the terrain potential – i.e. moving towards the obstacle)*

   (a) if $s =$ none then:

      i. if $\hat{c} \cdot \hat{d} < 0$ then pick s such that $\hat{d} \cdot \hat{s} \geq 0$

ii. else pick s such that $\hat{c} \cdot \hat{s} \geq 0$

(b) $\vec{a} \leftarrow \phi^2 \hat{d} + (1 - \phi^2)\hat{s}$

4. if $d < d_{min}$ then $\vec{a} \leftarrow b\hat{d} + (1 - b)\vec{a}$, where $b = \sqrt{d/d_{min}}$

   (this operations modulates the behavior after the agent gets within d_{min} of the obstacle)

5. at this point $\vec{a}$ contains the outcome of the decision process in the form of a direction of motion.

5.4 Experiments and Performance Analysis

In this section we present experimental results on the performance of the heuristic navigation algorithm and provide quantitative comparison with other approaches. The terrain we used is shown in Figure 5.1 (terrain dimensions are 300×300 units, terrain potential computed in 2000 iterations at 256×256 quantization, $d_{min} = 10$ units).

Table 5.2 provides experimental results for a set of 1000 randomly selected missions. A *mission* is defined by a pair of points which represent initial and final positions of an agent. For each mission, we run three different algorithms:

1. SHORTEST computes the optimal path from source to destination. It is based on an extension of Dijkstra's shortest path algorithm [Dij59] to continuous domain.

2. HEURISTIC is the algorithm we present in the previous section.

3. SIMPLE is equivalent to the GD Navigation behavior described in Chapter 3 - the agent always moves in a direction which will minimize the distance to destination (or gets stuck if

any action will actually increase the distance)

We chose to compare HEURISTIC with SHORTEST and SIMPLE because

- SHORTEST gives best results in terms of path length and success ratio but is computationally expensive and requires global terrain information,

- SIMPLE has very low computational complexity but very low success ratio too,

and HEURISTIC is an algorithm which tries to combine the best properties of both ends of the spectrum (i.e. high success ratio and low computational complexity).

The results in Table 5.2 contain three different columns labeled as Set 1, Set 2 and Set 3. These sets represent the domains of success for the three different algorithms:

- Set 1 contains all missions which are successfully solved by SHORTEST (i.e. all 1000 missions).

- Set 2 contains all missions which are successfully solved by HEURISTIC (974 missions).

- Set 3 contains all missions which are successfully solved by SIMPLE (339 missions).

We examine the performance of the algorithms on these sets in order to provide means to compare their performance within the respective domains - i.e. how well they perform when applied to easy or difficult missions.

One of the shortcomings of HEURISTIC is that it has trouble approaching the final destination when it is very close to an obstacle[4]. In order to analyze the effect of obstacle proximity we

[4]Step 4 in the algorithm description tries to minimize the effect of this problem by smoothly turning off obstacle avoidance when the agent is very near to the destination, but does not remove it completely.

examine the subsets of missions for which the destination potential ϕ is bound by a lower threshold ϕ_{min}. The rows in Table 5.2 provide results for ϕ_{min} ranging from 0 to 0.9 (i.e. we start with no limitations on proximity of the destination to obstacles and gradually eliminate missions for which the destination is deemed too close to an obstacle).

A close inspection of the results show that the success of HEURISTIC increases from 97.4% to 100% when ϕ at the destination is limited to 0.2 or higher and stays at 100%, while the success of SIMPLE varies between 34% and 40%. As can be seen, the improvement that HEURISTIC provides over SIMPLE is significant, especially when we take into consideration the fact that SIMPLE and HEURISTIC have the same computational complexity at runtime.

Figures 5.5 and 5.6 provide plots of the average path lengths from the Set 2 and Set 3 columns of Table 5.2. According to figure 5.5, HEURISTIC is able to find paths which are on average 20% worse than the optimal. In the case of trivial paths (i.e. paths that can be discovered by SIMPLE), HEURISTIC finds paths which are on average about 10% percent worse than optimal while paths discovered with SIMPLE are on average about 5% worse than optimal. The trouble experienced by HEURISTIC for destinations that are close to obstacle boundaries are apparent in the far left part of figure 5.6. Finally, figure 5.7 shows nine randomly selected missions from the 1000 used in the experiments.

5.5 Discussion

We introduced a novel approach to path-finding and navigation in an environment with complex obstacles based on the idea of transforming a real terrain into an obstacle-free virtual space. We also presented, as a proof of concept, a heuristic algorithm which, by incorporating terrain charac-

Table 5.2: Performance of SIMPLE, HEURISTIC and SHORTEST on the three sets for different ϕ_{min}

	Set 1	Set 2		Set 3					
ϕ_{min}	A_b	A_b	A_h	A_b	A_h	A_s	N_b	N_h	N_s
0	204.09	204.24	234.20	137.46	174.01	142.33	1000	974 (97.4%)	339 (33.9%)
0.05	205.21	204.98	233.75	141.90	154.50	146.93	928	921 (99.2%)	319 (34.3%)
0.1	204.92	204.96	232.33	142.63	155.38	147.64	883	879 (99.5%)	310 (35.1%)
0.15	204.03	204.07	230.89	143.33	155.13	148.36	850	846 (99.5%)	305 (35.8%)
0.2	205.44	205.44	232.23	144.86	156.84	150.00	814	814 (100%)	292 (35.8%)
0.3	206.49	206.49	231.63	145.08	157.70	150.10	727	727 (100%)	264 (36.3%)
0.4	207.22	207.22	232.42	147.74	159.90	152.97	657	657 (100%)	240 (36.5%)
0.5	209.30	209.30	234.08	149.75	162.44	155.25	559	559 (100%)	199 (35.5%)
0.6	209.74	209.74	234.57	153.93	165.91	159.64	487	487 (100%)	174 (35.7%)
0.7	209.68	209.68	233.86	151.69	162.13	157.09	404	404 (100%)	149 (36.8%)
0.8	212.47	212.47	237.87	155.45	167.42	161.01	310	310 (100%)	117 (37.7%)
0.9	210.65	210.65	233.64	162.06	172.90	167.38	196	196 (100%)	78 (39.7%)

teristics into the decision process is able to significantly improve the success ratio of reactive agent navigation methods and at the same time retain their low run-time computational requirements. This is achieved by processing information about terrain-specific features into a form which allows efficient fusion of this data into the decision process.

One of the advantages of the proposed way of abstracting relevant terrain information is that the amount of processed data depends only on the dimensions of the terrain at a certain level of detail and can be computed and accessed in a completely distributed manner. This information is only terrain-specific and does not depend on a particular mission. Therefore, it only needs to be computed once in order to support a large number of agents sharing the same environment. Due to its low run-time computational requirements, our approach is also well suited for control of autonomous robotic platforms with limited computational power.

Last but not least, we believe that the problem of goal-based navigation is closely related to the

Table 5.3: Explanation of the notation used in Table 5.2

Set 1	:	All missions for which $\phi \leq \phi_{min}$ at destination
Set 2	:	Subset of set 1 for which HEURISTIC was successful
Set 3	:	Subset of set 1 for which SIMPLE was successful
A_b	:	Average path length of shortest paths
A_h	:	Average path length for paths discovered by HEURISTIC
A_s	:	Average path length for paths discovered by SIMPLE
N_b	:	Number of missions for which $\phi \geq \phi_{min}$ at destination
N_h	:	Number of paths discovered (out of a total of N_b) by HEURISTIC
N_s	:	Number of paths discovered (out of a total of N_b) by SIMPLE

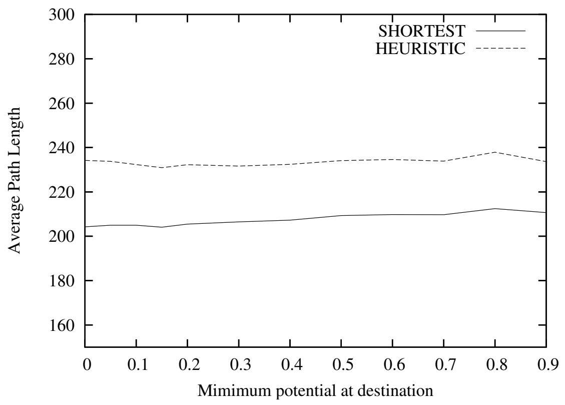


Figure 5.5: Average path length of set 2 missions w.r.t. ϕ_{min}

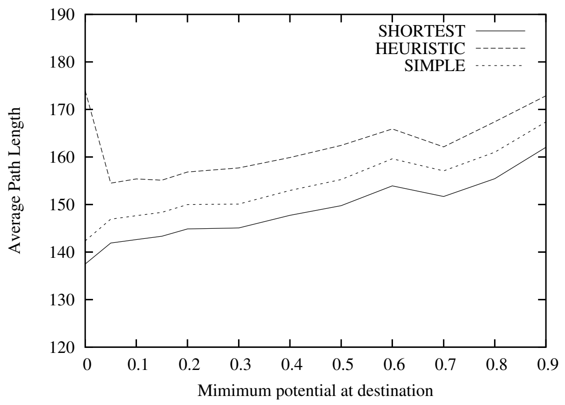
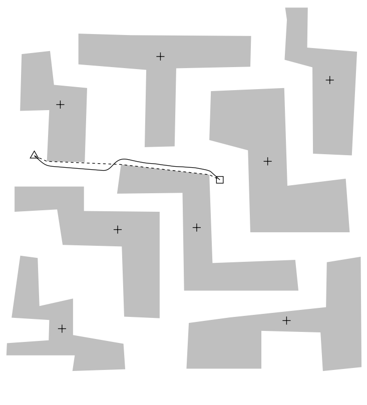
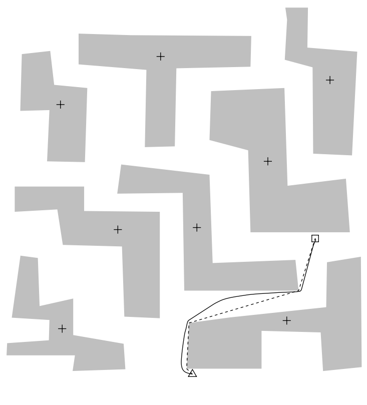
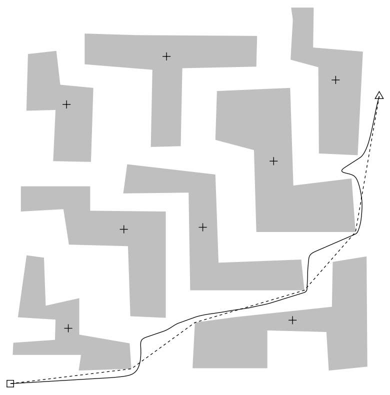


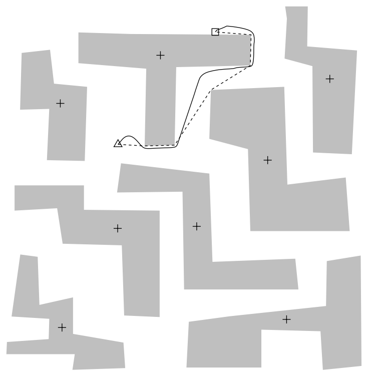
Figure 5.6: Average path length of set 3 missions w.r.t. ϕ_{min}

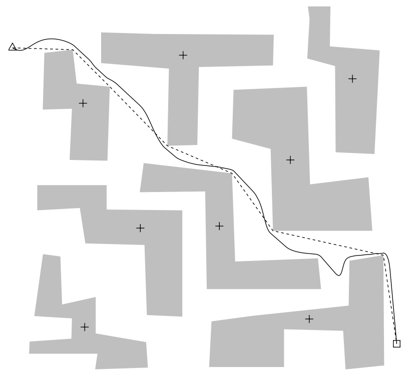
inverse problem of discovering goals of agents based on observed motion, and we hope that our approach will contribute to a better understanding of agent mobility in complex environments and situation awareness.

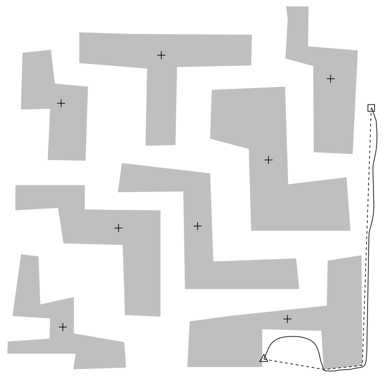

(a) mission #67

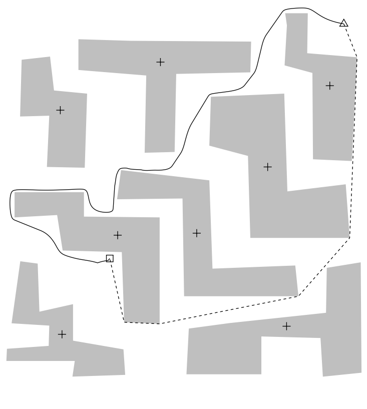

(b) mission #89

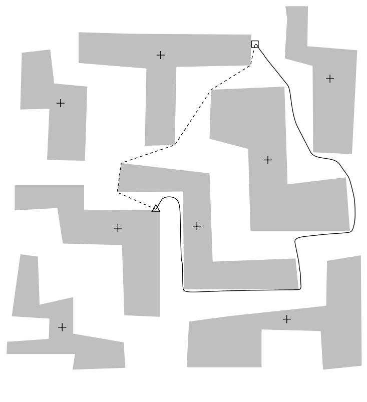

(c) mission #122


(d) mission #349


(e) mission #422


(f) mission #588


(g) mission #700


(h) mission #701

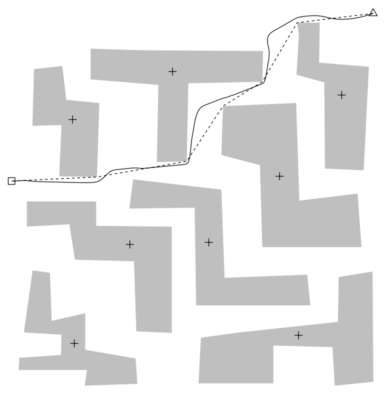

(i) mission #810

Figure 5.7: A random selection of mission outcomes. Initial and final positions are marked with a triangle and a square, respectively. Dotted lines represent SHORTEST paths and solid lines represent HEURISTIC paths.

CHAPTER 6

DISCOVERING AND TRACKING GROUPS

6.1 Introduction

Part of the discussion of experimental observations in Chapter 4 is dedicated on emphasizing the fact that the ability to detect unexpected emergent events may provide an important feedback mechanism for dealing with such situations. In the context of military simulations, a large class of such events are associated with the behavior of groups in a dynamic environment and, therefore, can be approached as the problem of detecting and tracking groups.

However, this is only one of the reasons for exploring this area and it is certainly not the most important one. The ability to detect and track groups of agents is considered a fundamental part of the higher-level data fusion model. Unfortunately, this is an area which traditionally has not received enough attention from the fusion community [Pan04]. As a direct consequence of the increase of deployment of high-level technology in modern military conflicts and the general shift from the platform-centric warfare model to a network-centric one, there has been a sharp increase in the importance of achieving complete situational awareness which is the ultimate goal of higher-level data and information fusion.

In the rest of the chapter, we will introduce two algorithms for a) detecting groups of agents, and b) track groups and detect events of interest.

6.2 Experimental Setup

The second experiment described in Chapter 4 provides a good example of emergent behavior. We will, therefore, utilize the same scenario as a test case for group detection and tracking. The basic configuration of the experiment is shown in Figure 4.2. There are three teams of agents (color-coded as Red, Green and Blue, 5 agents in each group, 15 agents in total), which have the following objectives:

- Blue team has to go the destination while avoiding the Red team.
- Red team has to intercept and destroy the Blue team while avoiding the Green team.
- Green team has to help the Blue team by trying to intercept and destroy the Red team.

The terrain also contains a number of simple obstacles (Trees, represented as green circles of varying radii).

6.2.1 Observing Emergent Behavior

Before introducing our approach for group detection and tracking, we will discuss a number of observations related to emergent situations arising in our test case which are particularly interesting in military scenarios. When running our simulations, we were able to observe all of the following:

- Spatially well-formed and distinct groups behaving in accordance with their instructions.
- Adversarial groups fusing into a single formation when they attack each other.
- Friendly groups which are difficult to distinguish from each other due to spatio-temporal proximity of their trajectories.

- Partial break-up of a group due to inability of some group members to “keep pace” with the rest of the group.

- Complete break-up of a group because of a dominant goal (i.e. urge to evade adversaries dominates other goals such as staying together, and as a result the group is completely dissolved).

We examine these events in detail with the hope that a close scrutiny will provide important insight into the nature of agent behavior and how raw observation data can be exploited best for differentiation of groups of related agents.

Let us first look at the spatial behavior of agents. We know that obstacles present in the terrain will influence the motion of agents. For this reason we examine the scenario both the normal terrain and in an environment without any obstacles. In both cases we look at how group formations develop as a function of time. Figure 6.1 shows four snapshots of the simulation at different times (0, 300, 600 and 900 time steps) when obstacles are present in the terrain. The obstacle-free versions are shown in Figure 6.2.

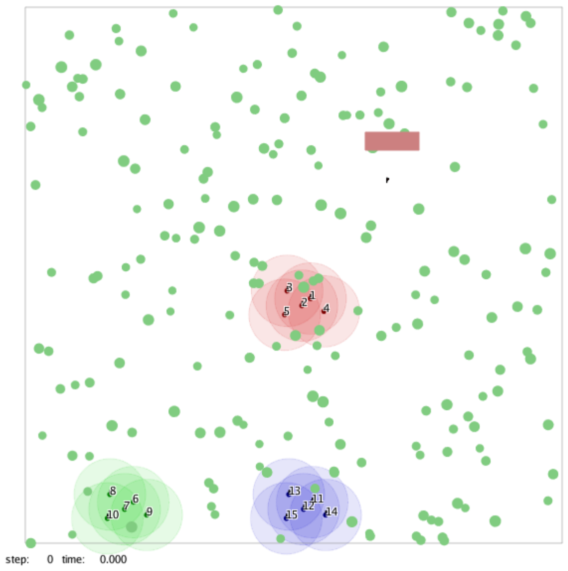
The evolution of the spatial configuration of the different teams throughout the simulations are shown as colored trails representing the trajectories of the agents. Each agent is also identified by a unique number (agents 1-5 are in the red team, 6-10 are in the green team and 11-15 are in the blue team).

An example of how terrain affects mobility is depicted in Figure 6.1(a). Agent 3 has trouble clearing some obstacles and eventually falls behind the group. As a result, it is able to avoid a confrontation with the green team and later engage the blue team while its teammates are mostly destroyed by the green team. This "change of plan" was not explicitly encoded in the mission objectives, but is an outcome of the complex interactions of different aspects of agent behavior. Another interested "side effect" is the chase going on between agents 9 and 2. Such unexpected emergent behavior is not observed in Figure 6.2, mostly due to the fact that there are no obstacles to interfere with the motion of groups. It is also interesting to note that the green and blue teams as shown in Figures 6.1(b) and 6.2(b) are too close to each other to be easily distinguished by looking at their spatial formation alone (assuming that an independent observer has no other information).

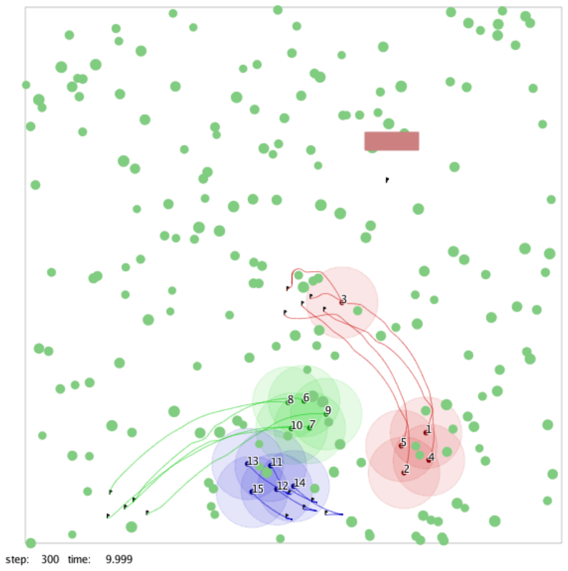
Besides spatial configuration, we also measure instantaneous agent behavior as a function of time. Within the setting of these simulations, instantaneous agent behavior refers to the direction in which an agent moves (i.e. velocity). A very good example of how these observations can be useful is the situation of the green and blue teams in Figures 6.1(b) and 6.2(b). While both teams are too close to each other for effective spatial discrimination, they can be easily identified as separate groups by observing that they happen to move in distinctively different directions. Measurement of the direction of motion throughout the whole simulation is shown in Figure 6.3 for four different configurations (i.e. in presence or absence of obstacles and weapons).

6.3 An MST-based Algorithm for Discovering Groups

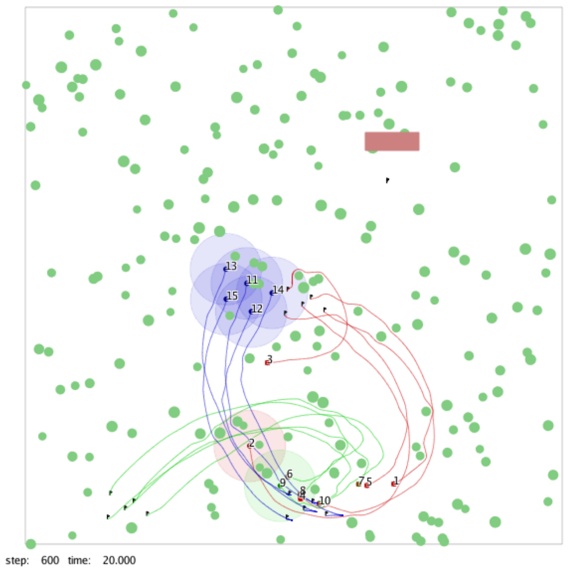
One of the problems of standard clustering methods is that they tend to work well when provided with large volume of raw data. Unfortunately, our scenarios involve relatively small number of agents that usually cannot provide statistical data of acceptable quality. Also, the dynamic nature of the groups that we are trying to detect makes this particular domain unsuitable for a range of clustering algorithms which depend on apriori information about the number of groups present



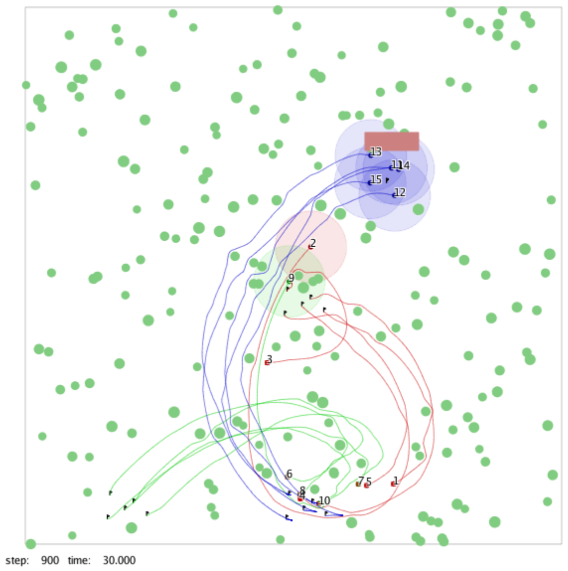
(a) steps=0



(b) steps=300

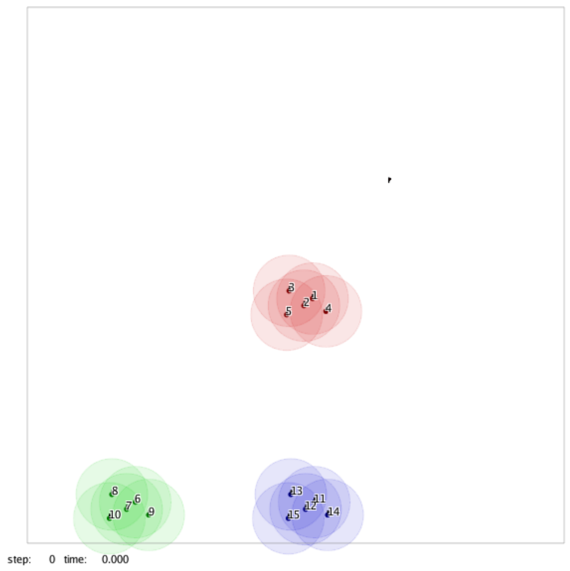


(c) steps=600

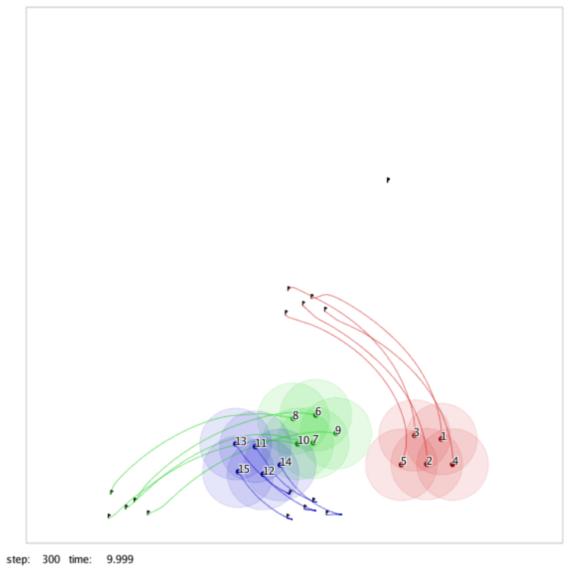


(d) steps=900

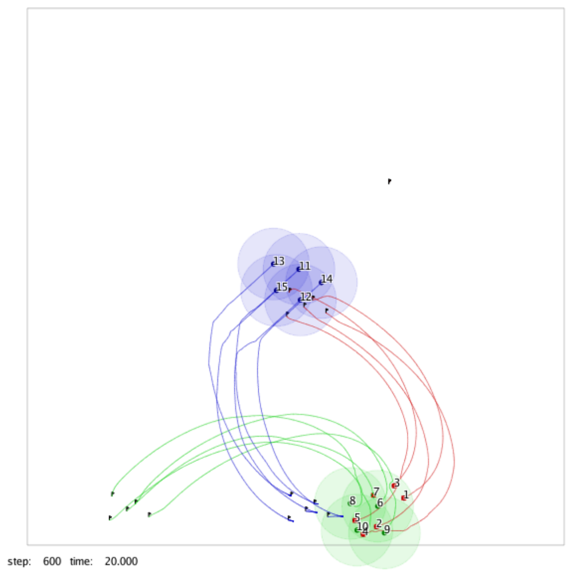
Figure 6.1: Evolution of the spatial configuration of the agents with respect to time in a terrain with obstacles



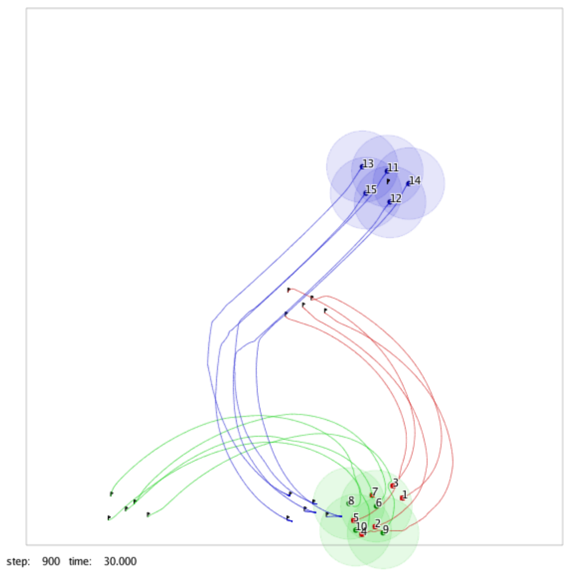
(a) steps=0



(b) steps=300

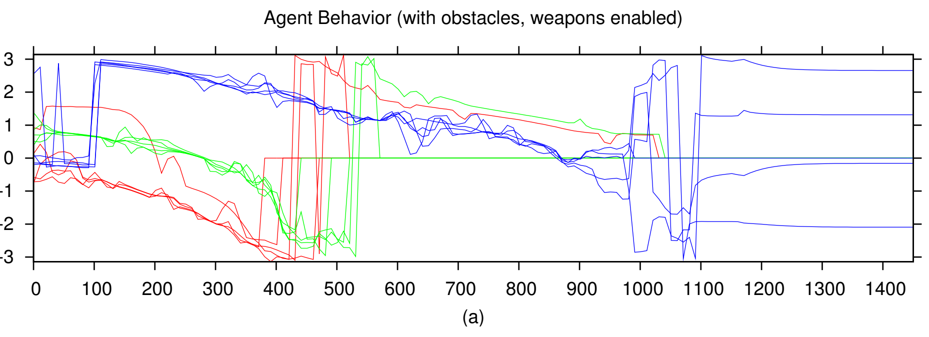


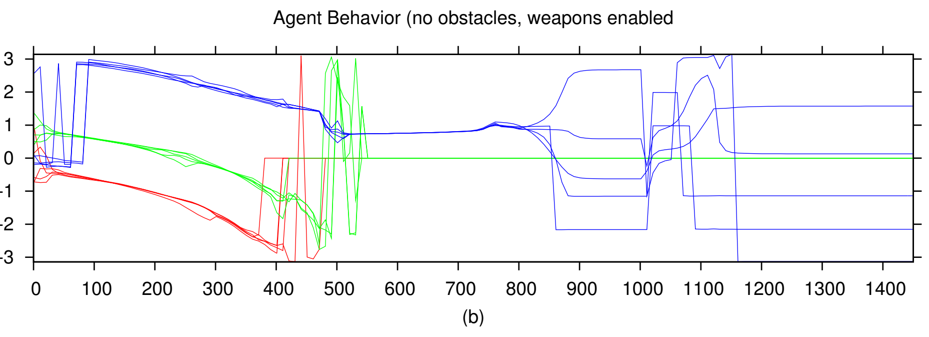
(c) steps=600

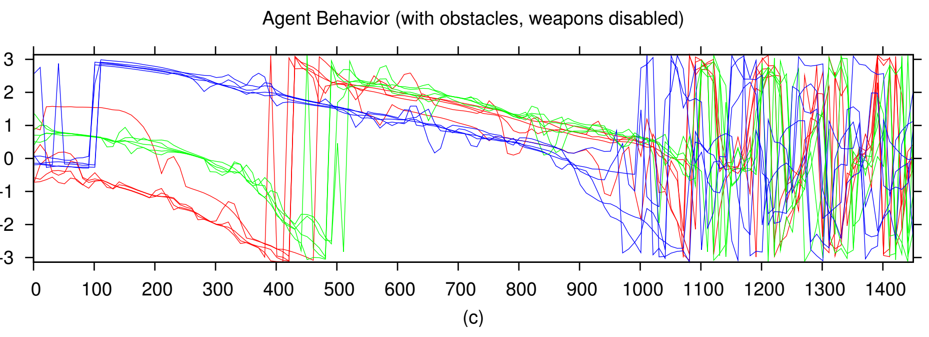


(d) steps=900

Figure 6.2: Evolution of the spatial configuration of the agents with respect to time in a terrain without obstacles







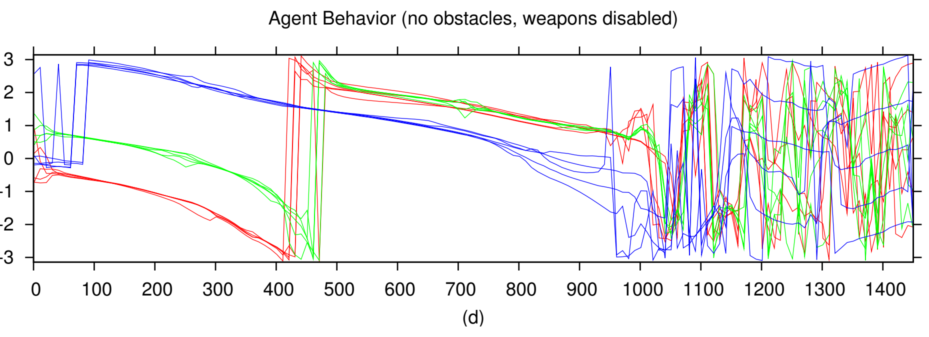


Figure 6.3: Evolution of the instantaneous agent behavior with respect to time. Vertical axis represents observed direction of motion (in radians). Horizontal axis represents simulation time steps.

(like K-means and derivatives).

In the process of researching alternative approaches we noticed that clustering algorithms which are designed specifically to address a particular class of problems tend to work much better than generic ones. This observation led to the development of a new clustering algorithm which is very efficient and performs particularly well for the class of scenarios that we are interested in. The description of our algorithm is as follows:

1. We assume that the environment contains a number of agents and that we can detect the number of agents (N) and their positions ($\vec{p}_i$) and/or their velocities ($\vec{v}_i$). N, $\vec{p}_i$'s and $\vec{v}_i$'s constitute the *input* to our algorithm.

2. A fully-connected bi-directional weighted graph with N vertices and $M = N * (N-1)/2$ edges is built such that vertices represent agents and the weights of the edges represent how well agents are related. In the case of spatial clustering, edge weights are the geometric distances ($|\vec{p}_i - \vec{p}_j|$) between agents, while in behavioral clustering the edge weights represent angular difference between the directions in which each agent moves - i.e. $\arcsin(\hat{v}_i \cdot \hat{v}_j)$, normalized to the range $(-\pi; +\pi]$.

3. The next step is to build a Minimum Spanning Tree (MST) for this graph. Only a few of the M edges will be part of the MST.

4. We then compute a density approximation of the distribution of MST edges. The particular method we use is Gaussian Parzen window estimation. This method has one configurable parameter σ, the value of which will depend on whether the algorithm works in spatial or behavioral domain.

5. An edge cut-off threshold ϵ is computed from the density estimation. This is done by first finding the highest peak in the density and then locating the smallest edge length (ϵ) on the right of the peak for which the density drops to half of the maximum.

6. The edges in the MST that have values above the cut-off threshold are removed. As a result of this operation, the MST is partitioned in a collection of sub-trees (in Graph Theory, such collections are better known as *forests*).

7. Each tree in this forest is assumed to represent a separate group of agents.

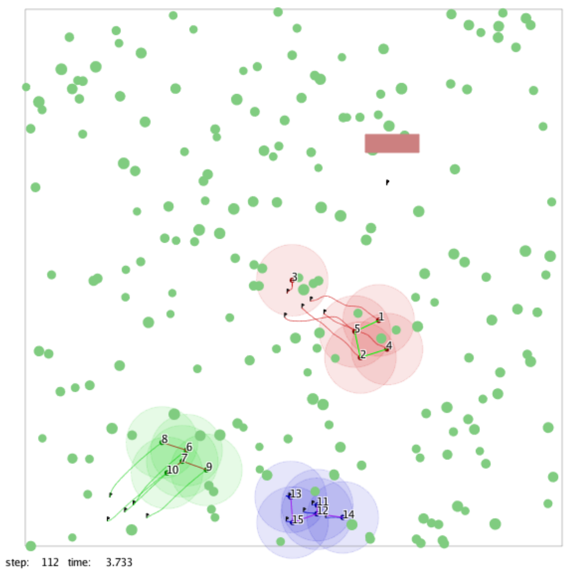
The operation of this algorithm is illustrated graphically in Figure 6.4.

6.4 An Algorithm for Temporal Analysis of Group Evolution

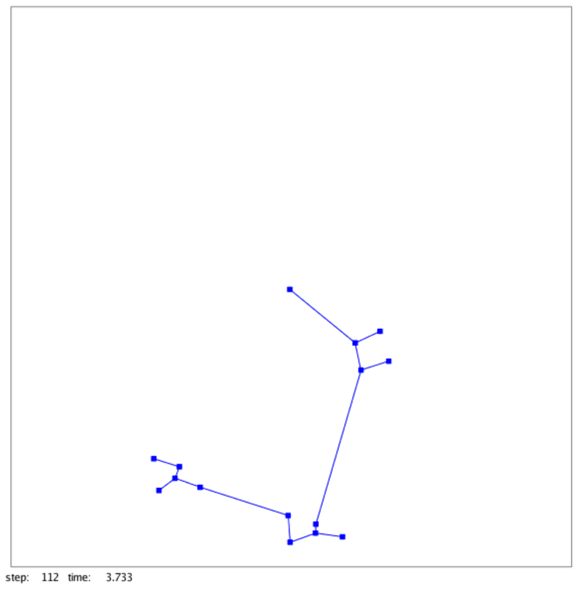
In this section we show how our method of clustering can be used as a valuable tool for detecting certain events of interest. To this end, we provide a novel algorithm which detects events by performing temporal analysis over the evolution of groups. The output of this algorithm is a sequence of triggers describing possible or confirmed events. The events indicate either the combination of a number of groups into a single group or the breakup of a single group into a number of distinct components.

The description of the algorithm is as follows:

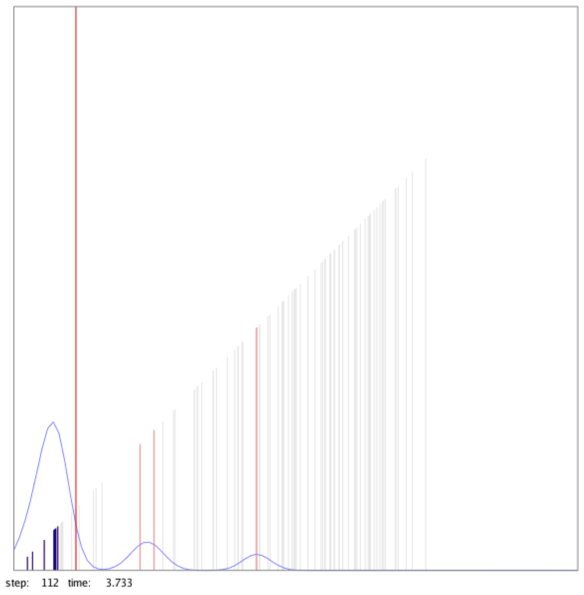
Throughout the simulation we keep a list of viable groups G that have been detected so far. G is initially empty. Each group in G has two counters associated with it: *age* and *duration*. *Age* represents the time difference between current simulation time and the most recent time the



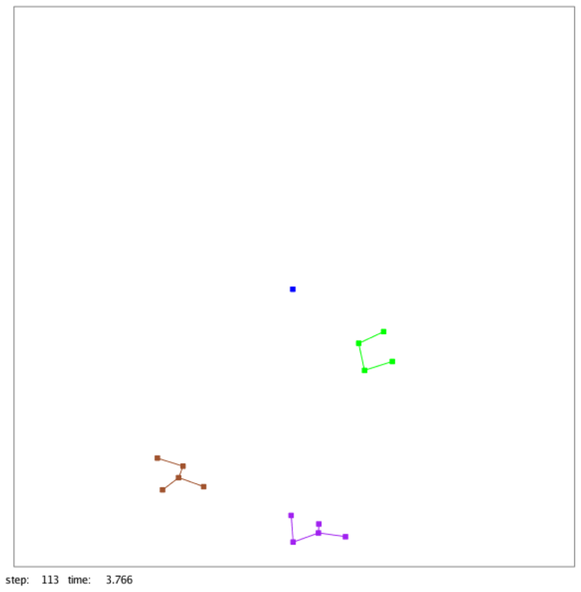
(a) Simulation Screenshot, 15 agents



(b) MST based on spatial metrics



(c) Histogram and density estimation of MST edge lengths ($\sigma = 5$). ϵ is marked with a red vertical line.



(d) Partial MST forest after removal of edges above the cut-off limit

Figure 6.4: Graphical illustration of the MST-based clustering algorithm

group was detected. *Duration* represents for how long the group was consistently detected at every timestep counting backwards from the current simulation time.

At each timestep, the following operations are performed in this particular order:

1. Since agents can die during a mission, we first update all the groups in G by removing agents that have been destroyed during the last step. If group duplications occur as a result of this removal, we combine the duplicates into a single group with a duration set to the maximum of both durations and an age set to the minimum of both ages. If a group becomes empty, it is removed from G.

2. At each time step, the algorithm also receives a list of current groups N as detected by the clustering algorithm described in the previous section. Each group in N is marked as *old* or *new* depending on whether it is present in G or not.

3. For each group c in N that is also present in G and can be represented as a union of other groups in G we trigger a "Possible Group Combination" event if duration of c is greater that t_1 and a "Confirmed Group Combination" event if the duration of c is greater that t_2. When a combination is confirmed, the constituent groups are removed from G. t_1 and t_2 depend on the time scale of simulations (in our experiments, $t_1 = 15$ and $t_2 = 30$).

4. For each group s in G which can be represented as a union of groups in N we trigger a "Possible Group Separation" if the age of s is greater than t_1 and a "Confirmed Group Separation" event if the age of s is greater than t_2. When a separation is confirmed, s is removed from G.

5. Groups marked as *old* have their age set to 1 and duration increased by 1. *New* groups have

their age and duration set to 1. Groups in G which are not present in N have have their age increased by 1 and their duration set to 0.

6.5 Experimental Results

In this section we illustrate how our clustering and event detection algorithms operate on the test scenario. The results are shown in Figures 6.5 through 6.9. Each of these figures contain a the following pieces of information:

1. Terrain and agents (left part of the figure)
2. Density estimation for spatial clustering (top-middle graph)
3. Density estimation for behavioral clustering (top-right graph)
4. Spatial clustering results (middle-center square)
5. Behavioral clustering results (middle-right square)
6. Event triggers (bottom-right area). These triggers are generated by applying temporal analysis *only* to the spatial clustering of the agents.

Figure 6.5 shows the state of the simulator at time $t = 0$. There are three spatially distinct groups but only one behaviorally distinct group (this is due to the fact that all agents are stationary). No events have been triggered.

Figure 6.6 shows the state of the simulator at time $t = 2.733$. There are four spatially distinct groups. Behavior clustering seems to be noisy. The separation of agent 3 from the red team has been been marked as possible at $t = 1.86$ and later confirmed at $t = 2.36$.

Figure 6.7 shows the state of the simulator at time $t = 15.533$. There are two spatially distinct groups. The combination of the red team and the green team has been marked as possible at time $t = 14.29$ and later confirmed at $t = 14.79$. Agent 3 from the red team has just established contact with the blue team (however, as can be seen in the next figure, it will be destroyed before the algorithm can trigger possible group combination). Note also that the behavior clustering is able to resolve the difference between the blue team and agent 3 as well as properly distinguish the green and red teams as they engage each other.

Figure 6.8 shows the state of the simulator at time $t = 26.966$. There are three spatially distinct groups: the blue team, agent 9 from the green team, and agent 2 from the red team. Behavioral clustering considers agents 2 and 9 to be one group (as they behave in a similar fashion). The fight between the red and green teams seems to have resulted in another recombination at $t = 18.66$ to be followed by a separation at $t = 19.66$.

Figure 6.9 shows the state of the simulator at time $t = 33.7$. There is only one spatially and behaviorally distinct group. This event has been marked as possible at time $t = 32.56$ and confirmed at $t = 33.06$.

As it can be seen from these examples, the results achieved by our approach are quite promising. The spatial clustering, in particular, performs quite well when agent groups are markedly distinct. Although less effective in general, behavioral clustering seems to be particularly good at distinguishing different agent groups when the spatial clustering fails due to group proximity.

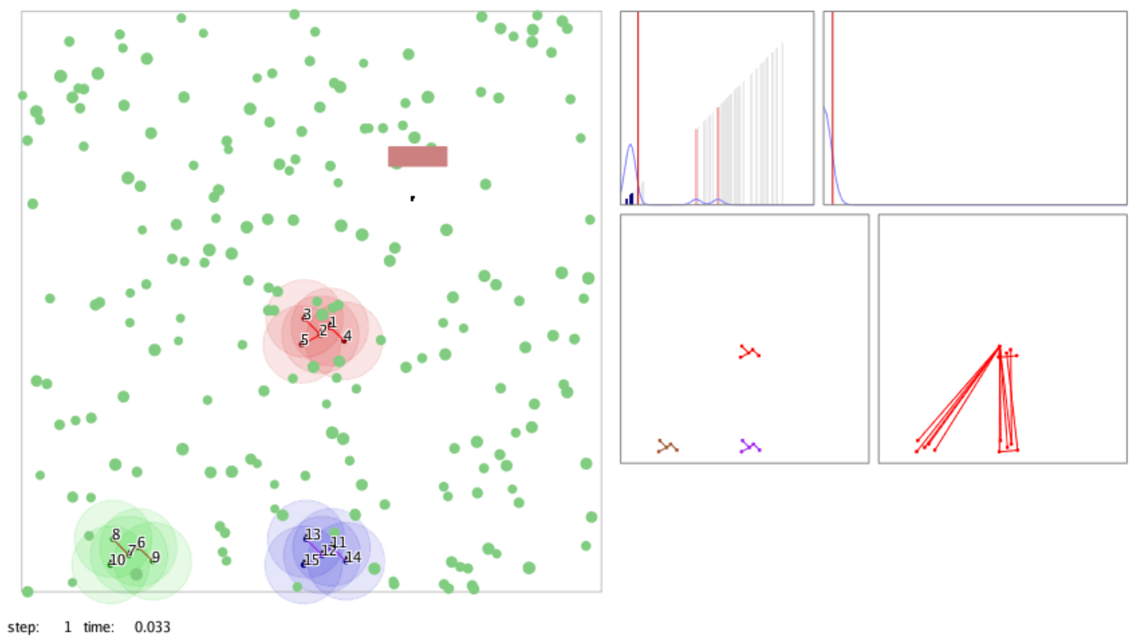
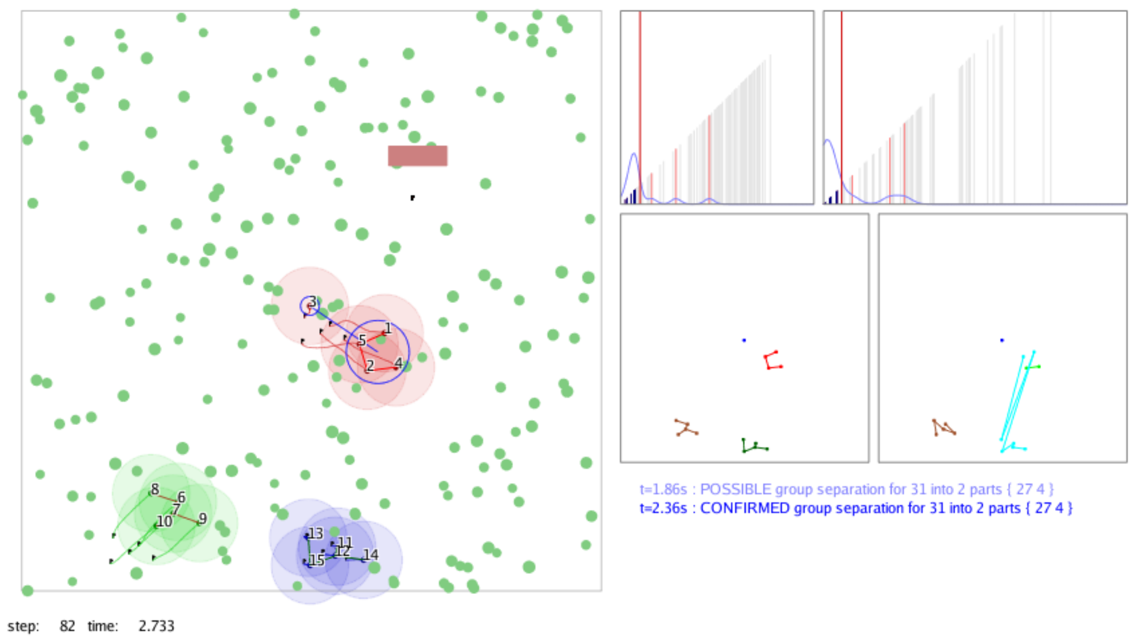
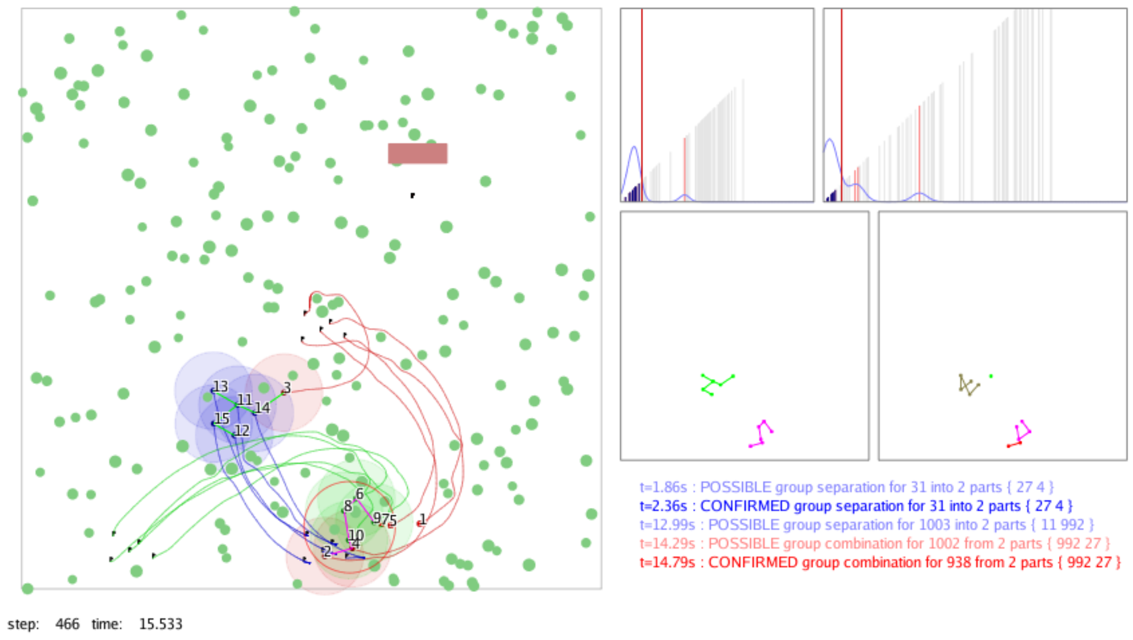


Figure 6.5: Spatial and Behavioral clustering example #1



```
t=1.86s : POSSIBLE group separation for 31 into 2 parts { 27 4 }
t=2.36s : CONFIRMED group separation for 31 into 2 parts { 27 4 }
```

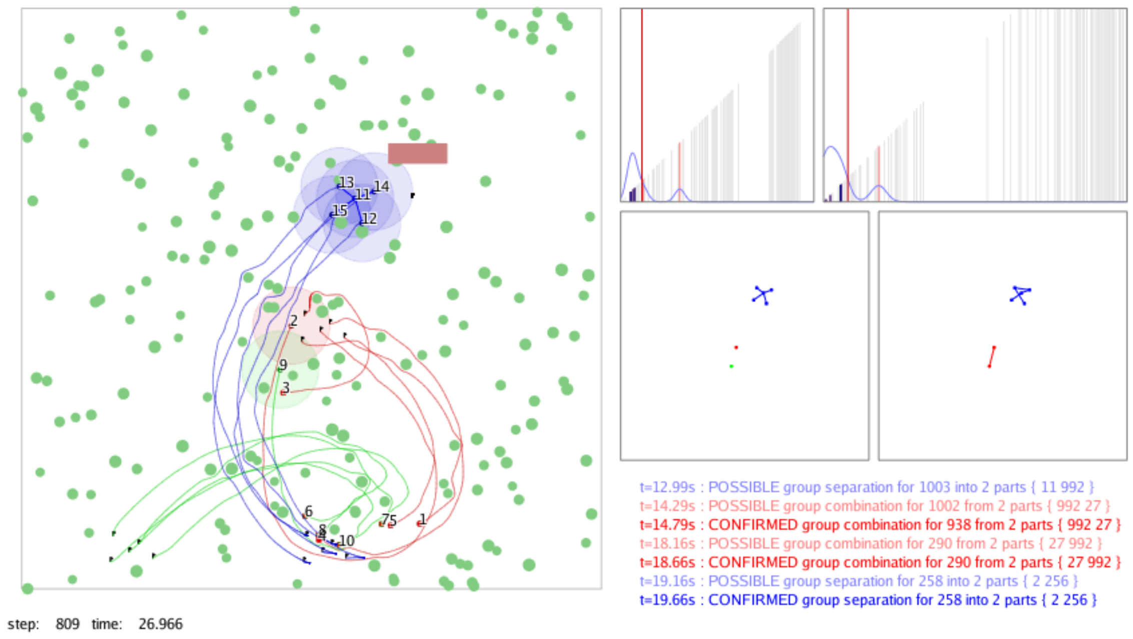
Figure 6.6: Spatial and Behavioral clustering example #2



```
t=1.86s : POSSIBLE group separation for 31 into 2 parts { 27 4 }
t=2.36s : CONFIRMED group separation for 31 into 2 parts { 27 4 }
t=12.99s : POSSIBLE group separation for 1003 into 2 parts { 11 992 }
t=14.29s : POSSIBLE group combination for 1002 from 2 parts { 992 27 }
t=14.79s : CONFIRMED group combination for 938 from 2 parts { 992 27 }
```

```
step:   466   time:   15.533
```

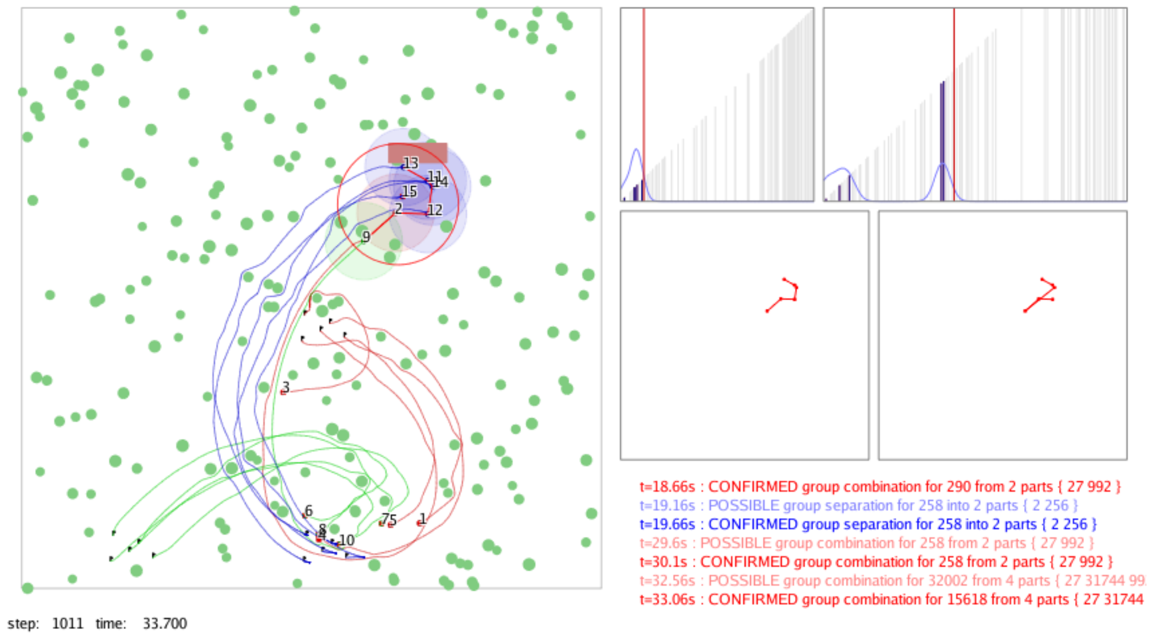
Figure 6.7: Spatial and Behavioral clustering example #3



```
t=12.99s : POSSIBLE group separation for 1003 into 2 parts { 11 992 }
t=14.29s : POSSIBLE group combination for 1002 from 2 parts { 992 27 }
t=14.79s : CONFIRMED group combination for 938 from 2 parts { 992 27 }
t=18.16s : POSSIBLE group combination for 290 from 2 parts { 27 992 }
t=18.66s : CONFIRMED group combination for 290 from 2 parts { 27 992 }
t=19.16s : POSSIBLE group separation for 258 into 2 parts { 2 256 }
t=19.66s : CONFIRMED group separation for 258 into 2 parts { 2 256 }
```

```
step:   809   time:   26.966
```

Figure 6.8: Spatial and Behavioral clustering example #4



```
t=18.66s : CONFIRMED group combination for 290 from 2 parts { 27 992 }
t=19.16s : POSSIBLE group separation for 258 into 2 parts { 2 256 }
t=19.66s : CONFIRMED group separation for 258 into 2 parts { 2 256 }
t=29.6s : POSSIBLE group combination for 258 from 2 parts { 27 992 }
t=30.1s : CONFIRMED group combination for 258 from 2 parts { 27 992 }
t=32.56s : POSSIBLE group combination for 32002 from 4 parts { 27 31744 99
t=33.06s : CONFIRMED group combination for 15618 from 4 parts { 27 31744
```

Figure 6.9: Spatial and Behavioral clustering example #5

6.6 Discussion

In this Chapter we presented two novel algorithms for detection and tracking of groups of agents and group-related events of interest. The first algorithm detects groups of agents by applying graph-theoretic methods on measures of spatial and behavioral similarity between agents. The second algorithm tracks groups and detects events related to inter-group dynamics by applying temporal analysis on the output of the first algorithm. Although relatively simple, our approach exhibits very promising performance as can be seen from the experimental results.

While the problem of tracking individual agents has been studied extensively as part of low-level data fusion and is considered a mature subject now, the problem of tracking groups and analyzing their dynamics has received very little attention until recently. We hope that our contributions to this area will have a positive impact on the field of higher-level data and information fusion as part of the efforts of improving situation awareness in modern military conflicts.

CHAPTER 7

CONCLUSION AND FUTURE RESEARCH DIRECTIONS

7.1 Contributions

In this thesis we investigate the problem of modeling teams of collaborating and competing autonomous agents in the context of simulations of military operations. We focus in particular on agent control methods which can be deployed in real-time as part of an augmented reality embedded training system.

The contributions presented in our work can be summarized as follows:

1. We present a behavior-based agent model which illustrates how simple reactive behavior mechanisms which model aspects of agent actions like motion towards a goal, group formation, etc., can be combined in order to achieve a variety of emergent behaviors which commonly arise in military conflict situations.

2. We also present an algorithm for navigation in terrains with non-convex obstacles based on fusing terrain information into a reactive decision process. Due to its very low runtime computational requirements, high success rate and excellent scalability, this algorithm is particularly well suited for simulating a large number of agents (civilians, soldiers or autonomous robotic platforms) in an urban combat environment.

3. We present an algorithm for detecting teams of agents based on applying graph-theoretical methods for analyzing spatial and behavior similarities.

4. We present an algorithm for detecting events of interest based on a temporal analysis of group dynamics.

7.2 Limitations

Throughout our research we tried to keep the type of the agents, terrain and the nature of adversarial actions that we consider as general as possible. There are two reasons for this:

1. The primary purpose of our research has been to produce novel ideas on how to solve some fundamental problems in this area rather than to focus narrowly on a more specific sub-problem. This decision was made with the hope to produce results which are applicable to a broader range of areas and therefore increase the significance of our contributions.

2. Besides the danger of limiting the impact of the achieved results, building a scenario with components which are realistic enough to be considered suitable for direct application by the military usually requires access to classified information about technological capabilities or expert know-how.

As a direct result, our agent, terrain and mission models are not directly applicable for simulation of scenarios which require a high-level of detail or realism.

When applied to real terrains, our navigation approach requires the ability to accurately measure obstacles at a relatively high resolution. If this is done in a centralized manner, the process of dissemination of this information can introduce a single point of failure. In its current form, our approach is based on the assumption that all free space in the terrain has a uniform cost of travel. This may not be the case, for example when significant differences in elevation are present.

Finally, the problems which our algorithm experiences when approaching a destination very close to an obstacle require further consideration.

The success of our approach to group detection and tracking is vitally dependent on the ability to detect and consistently identify all agents in an environment in an ideal manner. In their current form, our algorithms do not provide any means to deal with input from lower-level tracking methods which may occasionally provide inconsistent or incomplete information.

7.3 Future Research Directions

Our research results provide a number of possible ways for future advances.

As far as the agent model is concerned, we only used static behavior configuration within the examples presented in this thesis. Our implementation allows for dynamic change of the weighting parameters or even the wiring scheme of behaviors throughout a simulation[1]. This property can be used in conjunction with measurements of performance metrics in order to explore possible ways of improving performance through learning.

Both the navigation and tracking algorithms, as described, include components which are more proofs of concepts than matured designs. As such, we feel that they provide ample space for improvement.

Another direction of future research is exploring the possibility of applying our navigation approach within a different domain. For example, the duality between the problems of path-finding and routing of packets in networks implies that it may be possible to transform or adapt our ap-

[1]We have used dynamic agent behavior to model the transformation of a civilians into a terrorists as part of supporting other research within our group. However, this research is not described here.

proach for delivery-guaranteed routing in arbitrary-topology geographical networks without the need to use a routing table or an expensive search mechanism.

APPENDIX A

THE RANDOM NEURAL NETWORK

The random neural network (RNN) model, introduced by Gelenbe [Gel89, Gel90, Gel93], is an analytically tractable spiked neural network model which has been studied extensively all around the world in the past decade. Although it is based on non-linear mathematics, the mathematical structure of RNN is akin to that of queuing networks and it has "product form" just like many useful queuing network models. The applications that can use RNN cover almost all the areas in which typical neural networks are employed. Applications reported in the most recent RNN literature include, but are not limited to, associative memory, combinatorial optimization, texture generation, image processing, still image and video compression, function approximation, magnetic resonance imaging, pattern recognition, detection and classification of synchronous recurrent transient signals, mine detection, automatic target recognition, and computer communications [BK00, GXS99]. A hardware implementation of the RNN for a single neuron using TTL IC is proposed in 1996 and a chip design for a network of 16 neurons using CMOS technology is proposed in [BHA97].

Like other artificial neural network models, the work on the RNN model was initially motivated by the behavior of natural neural networks. The interconnection and interaction among the RNN neurons make it closer to biophysical reality than widely used artificial neuron models in which signals are represented by fixed signal levels. The RNN model is based on probabilistic assumptions and belongs to the family of Markovian queuing networks. The novelty with respect to usual queuing models lies in the concept of requests for removing work (negative customers) in addition to classical requests for performing work (positive customers). This novel class of models is referred to as G-network [GP98, LS97]. The significant feature of the model is that it is analytically solvable, and therefore computationally efficient, since its application is reduced to obtaining solutions to a system of a fixed-point equations [LS00]. In the RNN model, the rich mathematical structure of the network in terms of an infinite set of Chapman-Kolmogorov equations leads to a compact closed form solution for the feed-forward and recurrent case [GML99]. The RNN model accepts a product form solution, i.e., the network's stationary probability distribution can be written as the product of the marginal probabilities of the state of each neuron.

Signals in the Random Neural Network travel form of impulses with a unit amplitude. Signals can be positive or negative. Positive signals represent excitation and negative signals represent inhibition to the neuron receiving the signal. Each neuron i in the network has at time t an associated state $k_i(t)$ which is called its potential and is represented by a non-negative integer number.

When the potential of neuron i is positive, it is referred as being 'excited'. Excited neurons transmit (fire) signals at a Poisson rate r_i to other neurons or to the outside of the network. The transmitted signals will arrive as excitation signals at neuron j with probability p^+_{ij} and as inhibitory signals with probability p^-_{ij}. A neuron's transmitted signal can also leave the network with probability d_i, where

$$d_i = 1 - \sum_{j=1}^{n} (p^+_{ij} + p^-_{ij})$$

It is sometimes easier to work with firing rates rather than with probabilities. Let us define $w^+_{ij} = r_i p^+_{ij}$ as the rate of firing excitatory signals from neuron i to neuron j and $w^-_{ij} = r_i p^-_{ij}$ as the rate of firing inhibitory signals from neuron i to neuron j. The w matrices can be viewed as being analogous to the synaptic weights in classical neural networks, though they actually represent the rates of excitatory and inhibitory signal emission. Since the w matrices are formed as a product of rates and probabilities, they are guaranteed to be non-negative.

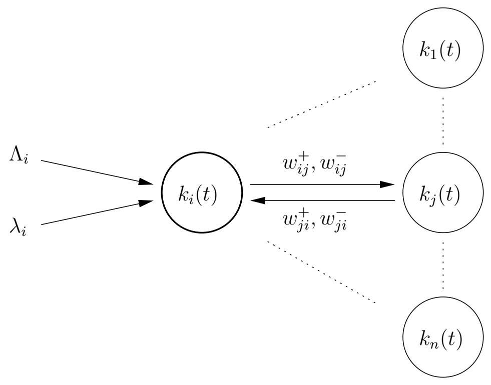


Figure A.1: Representation of a neuron in the RNN

Besides receiving signals from other neurons, each neuron i can also receive exogenous excitatory and inhibitory signals at Poisson rates of Λ_i and λ_i respectively.

Figure A.1 shows the representation of a neuron in the RNN using the model parameters that have been defined so far.

The dynamics of the RNN model is defined as follows:

- The potential $k_i(t)$ of a neuron will decrease by one whenever it fires a signal regardless of the type of the emitted signal.
- The potential $k_i(t)$ of a neuron will decrease by one whenever it receives an inhibitory signal from another neuron or from the exogenous inhibitory signal source. The potential will not decrease further if it is already zero.
- The potential $k_i(t)$ of a neuron will increase by one whenever it receives an excitatory signal from another neuron or from the exogenous excitatory signal source.
- The potential $k_i(t)$ will stay the same if none of the events described above happen.

Let $\widehat{k}(t) = \langle k_1(t), \ldots, k_n(t) \rangle$ be the vector of signal potentials at time t, and $\widehat{k} = \langle k_1, \ldots, k_n \rangle$

be a particular value of the vector. For the steady state analysis, let $p(\widehat{k})$ denote the stationary probability distribution which is given by $p(\widehat{k}) = \lim_{t\to\infty} Prob[\widehat{k}(t) = \widehat{k}]$ if it exists. Thus, in steady state, $p(\widehat{k})$ must satisfy the global balance equations:

$$\begin{aligned}
& p(\widehat{k}) \sum_i [\Lambda(i) + [\lambda(i) + r(i)]\mathbf{1}[k_i > 0]] \\
= & \sum_i [p(\widehat{k}_i^{+}) r(i) d(i) + p(\widehat{k}_i^{-}) \Lambda(i) \mathbf{1}[k_i > 0] \\
+ & p(\widehat{k}_i^{+}) \lambda(i) + \sum_j \{ p(\widehat{k}_{ij}^{+-}) r(i) p^{+}(i,j) \mathbf{1}[k_j > 0] \\
+ & p(\widehat{k}_{ij}^{++}) r(i) p^{-}(i,j) + p(\widehat{k}_i^{+}) r(i) p^{-}(i,j) \mathbf{1}[k_j = 0] \}]
\end{aligned}$$

where the vectors used are

$$\begin{aligned}
\widehat{k}_i^{+} &= \langle k_1, \ldots, k_i + 1, \ldots, k_n \rangle \\
\widehat{k}_i^{-} &= \langle k_1, \ldots, k_i - 1, \ldots, k_n \rangle \\
\widehat{k}_{ij}^{+-} &= \langle k_1, \ldots, k_i + 1, \ldots, k_j - 1, \ldots, k_n \rangle \\
\widehat{k}_{ij}^{++} &= \langle k_1, \ldots, k_i + 1, \ldots, k_j + 1, \ldots, k_n \rangle.
\end{aligned}$$

Let q_i be the steady state probability that neuron i is excited, that is,

$$q_i = \lim_{t\to\infty} Prob[k_i(t) > 0], \;\; i = 1, \ldots, n.$$

It was shown in [Gel89] that,

$$q_i = \lambda^{+}(i)/[r(i) + \lambda^{-}(i)], \tag{A.1}$$

where the $\lambda^+(i)$, $\lambda^-(i)$ for $i = 1, \dots, n$ satisfy the system of non-linear simultaneous equations

$$\begin{aligned} \lambda^+(i) &= \Lambda(i) + \sum_j q_j r(j) p^+(j,i), && \text{(A.2)} \\ \lambda^-(i) &= \lambda(i) + \sum_j q_j r(j) p^-(j,i). && \text{(A.3)} \end{aligned}$$

Clearly, $\lambda^+(i)$ and $\lambda^-(i)$ are the arrival rate of the positive and negative signals to neuron i, therefore, q_i is equal to the ratio of the sum of all the rates of arriving positive signals to the sum of the rates of arriving negative signals together with the firing rate of neuron i. If a unique non-negative solution $\{\lambda^+(i), \lambda^-(i)\}$ exists to the above equations such that $q_i < 1$, then the network stationary probability distribution has the form

$$p(\widehat{k}) = \prod_{i=1}^{n} [1 - q_i] q_i^{k_i}.$$

For a network with n neurons, the network parameters are two n by n "weight matrices" $\mathbf{W}^+ = \{w_{ij}^+\}$ and $\mathbf{W}^- = \{w_{ij}^-\}$ which need to be learned from input data through learning algorithms.

APPENDIX B

REINFORCEMENT LEARNING IN RNN

Various techniques for learning may be applied to the RNN. These include Hebbian learning, gradient based learning, and reinforcement learning. In this paper we use Reinforcement Learning based on the RNN model [Hal99] which was originally suggested for navigation in a maze. In this appendix, we are going to describe Reinforcement Learning and its adoption for Random Neural Network model in general and in our specific environment, which is terrain navigation.

One of the major classes of stimulus-response learning is instrumental conditioning (also called operant conditioning) where the organism is allowed to have an active role in the learning situation. An organism is allowed to adjust its behavior according to the consequences of that behavior. That is, when a behavior is followed by favorable consequences, such behavior tends to occur more frequently; but when it is followed by unfavorable consequences, it tends to occur less frequently. Collectively, favorable consequences are referred to as *reward* and unfavorable consequences are referred to as *punishment*.

Extinction is a well-known property in animal learning, and it is extremely important to the survival of living organism because it provides adaptation to changing conditions. Extinction refers to the decline of non-reinforced or punished association between some actions and their responses. It allows the system to learn new associations after forgetting those ones that are no longer valid, and prevents it from learning further associations based on those that are not valid and from getting stuck to previously learned actions.

In an artificial neural network, including RNN, it is the weight values assigned to the corresponding interconnections that allow the network to learn, to remember, and to react to the environment. Reinforcement Learning is one of the ways to build and update these weight values. It is similar to supervised training except that, instead of being given the correct output at each individual training trial, the network receives only a grade (could be either a *reward* if the grade is high, or a *punishment* otherwise) that tells it how well it has done over a sequence of training trials. Therefore, the learner in reinforcement learning is not told which action to take, but instead must discover by itself which actions yield the highest reward by trying them.

In [Hal97], a reinforcement function is provided for RNN along with the linear-reward-only weight update rule to solve the maze learning problem. It performs well for the stationary environment, but it suffers from getting stuck in the previously remembered actions and therefore losing its ability to adapt in changing conditions. In order to make further exploration possible for the same problem, [Hal99] proposes a reinforcement learning scheme for RNN which takes the "internal expectation of reinforcement" into consideration in such a way that it behaves as reward learning as long as the reward for the learned action is not below the expectation and it behaves as punishment learning otherwise, so that other possibilities are explored. Both of the above two approaches (in fact, the second is an extension to the first one) set all the negative weights p_{ij}^- to 0 and remain so, therefore, they don't fully use of the structure of RNN.

In the case of navigation, a Random Neural Network is used both for storing the weights and making decisions. The weights of the network are updated so that decisions are reinforced or weakened depending on whether they have been observed to contribute to increasing or decreasing the accomplishment of the declared goal.

Given some Goal G that the agent has to achieve as a function to be minimized (e.g. expected time to reach a destination or probability of getting killed, or a combination of the two), we formulate a reward R which is proportional to G^{-1}. Successive measured values of the R are denoted by R_l, $l = 1, 2,$ .. These are first used to compute a decision threshold:

$$T_l = aT_{l-1} + (1 - a)R_l, \tag{B.1}$$

where a is some constant $0 < a < 1$, typically close to 1.

A fully interconnected RNN with as many neurons as the decision outcomes is constructed. In the case of navigation the agent can decide to go to any of the 8 neighboring cells, therefore an 8-neuron decision network will be constructed. The weights will be

Let the neurons be numbered $1, ..., n$. Thus for any decision i, there is some neuron i. Decisions in this Reinforcement Learning algorithm with the RNN are taken by selecting the decision j for which the corresponding neuron is the most excited, i.e. the one which has the largest value of q_j. Note that the l^{th} decision may not have contributed directly to the l^{th} observed reward because of time delays between cause and effect.

Suppose that we have now taken the l^{th} decision which corresponds to neuron j, and that we have measured the l^{th} reward R_l. Let us denote by r_i the firing rates of the neurons before the update takes place.

What we do is first to determine whether the most recent value of the reward is larger than the previous "smoothed" value of the reward which we call the threshold or internal expectation T_{l-1}. If that is the case, then we increase very significantly the excitatory weights going into the neuron that was the previous winner (in order to reward it for its new success), and make a small increase of the inhibitory weights leading to other neurons. If the new reward is not better than the previously observed smoothed reward (the threshold), then we simply increase moderately all excitatory weights leading to all neurons, except for the previous winner, and increase significantly the inhibitory weights leading to the previous winning neuron (in order to punish it for not being very successful this time). This is detailed in the algorithm given below.

We first compute T_{l-1} and then update the network weights as follows for all neurons i. Suppose j was the previous winner:

- If $T_{l-1} \quad \leq \quad R_l$
  - $w_{ij}^{+} \leftarrow w_{ij}^{+} + (R_l - T_{l-1})$,
  - $w_{ik}^{-} \leftarrow w_{ik}^{-} + \frac{(R_l - T_{l-1})}{n-1}$, $\forall\, k \neq j$.
- Else
  - $w_{ik}^{+} \leftarrow w_{ik}^{+} + \frac{(T_{l-1} - R_l)}{n-1}$, $\forall\, k \neq j$,
  - $w_{ij}^{-} \leftarrow w_{ij}^{-} + (T_{l-1} - R_l)$.

Then we re-calculate all the weights by carrying out the following operations, to avoid obtaining weights which indefinitely increase in size. First for each i we compute:

$$r_i^* = \sum_{m=1}^{n} [w_{im}^{+} + w_{im}^{-}], \tag{B.2}$$

and then re-calculate the weights with:

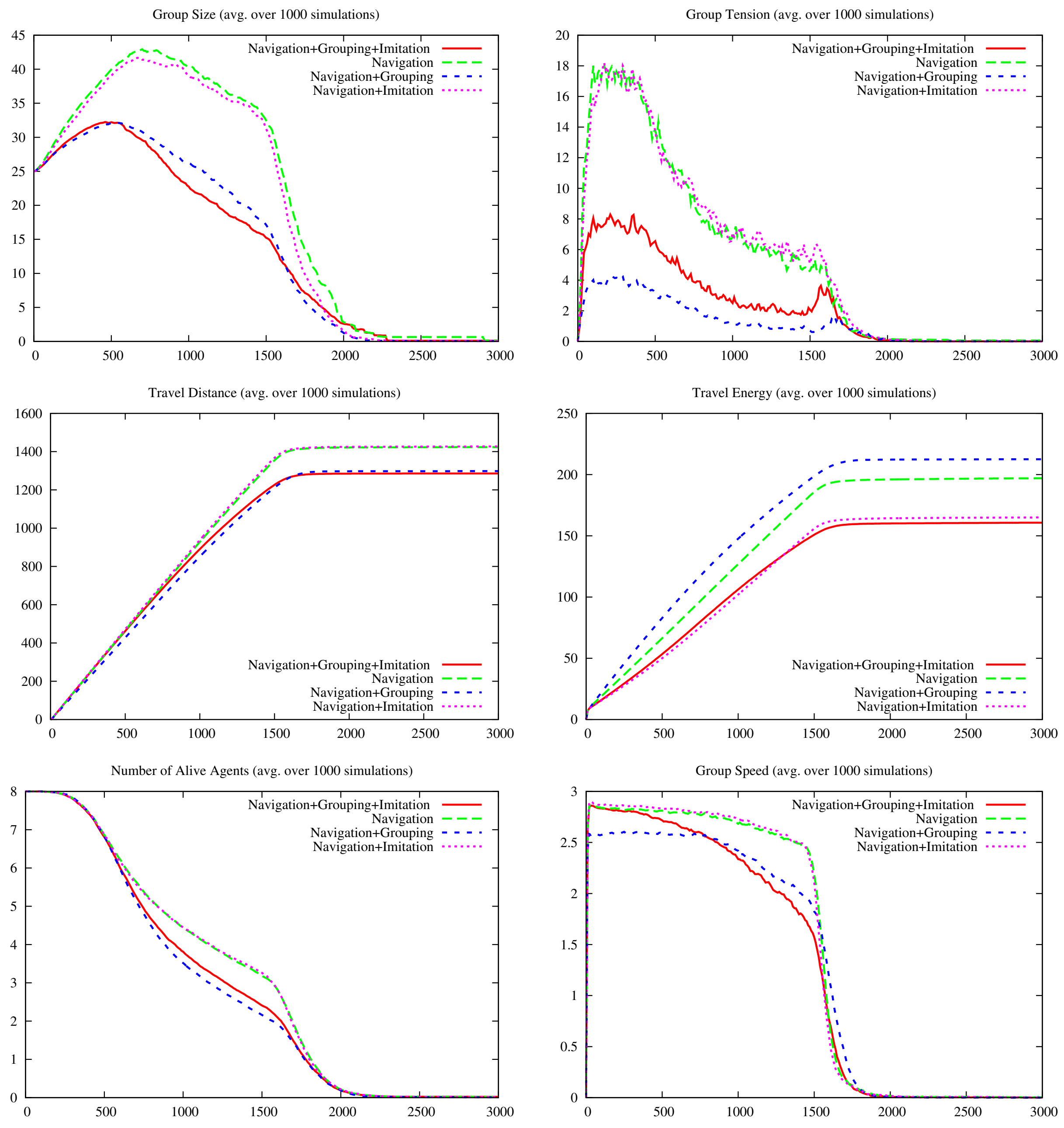
$$w_{ij}^{+} \leftarrow w_{ij}^{+} * \frac{r_i}{r_i^*},$$

$$w_{ij}^{-} \leftarrow w_{ij}^{-} * \frac{r_i}{r_i^*}\ .$$

Finally, the probabilities q_i are computed using the non-linear equations (A.1), (A.2), and (A.3) with fixed-point iterations, which lead to a new decision based on the neuron with the highest probability of being excited.

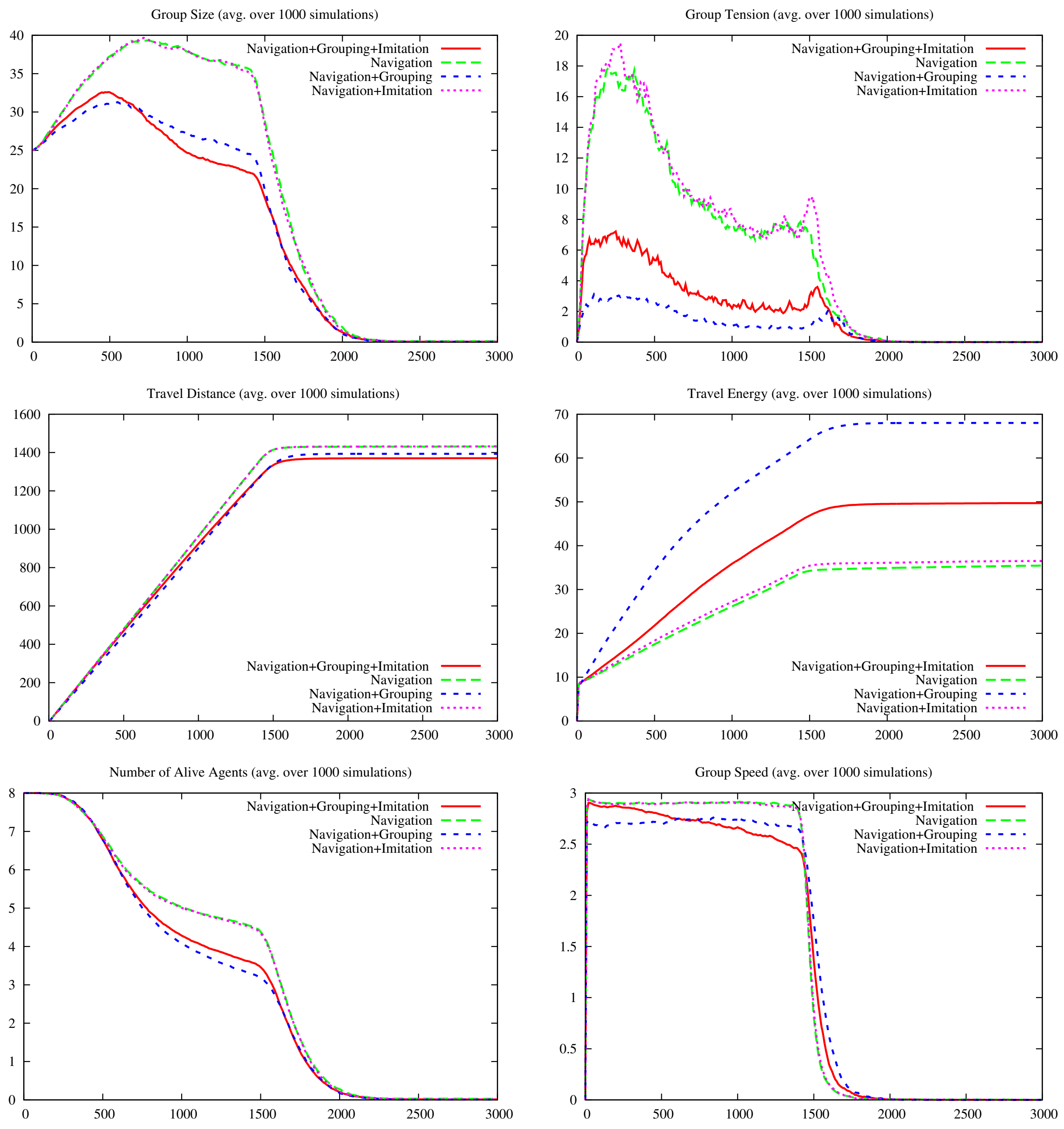
APPENDIX C

DETAILED RESULTS FOR EXPERIMENT 1



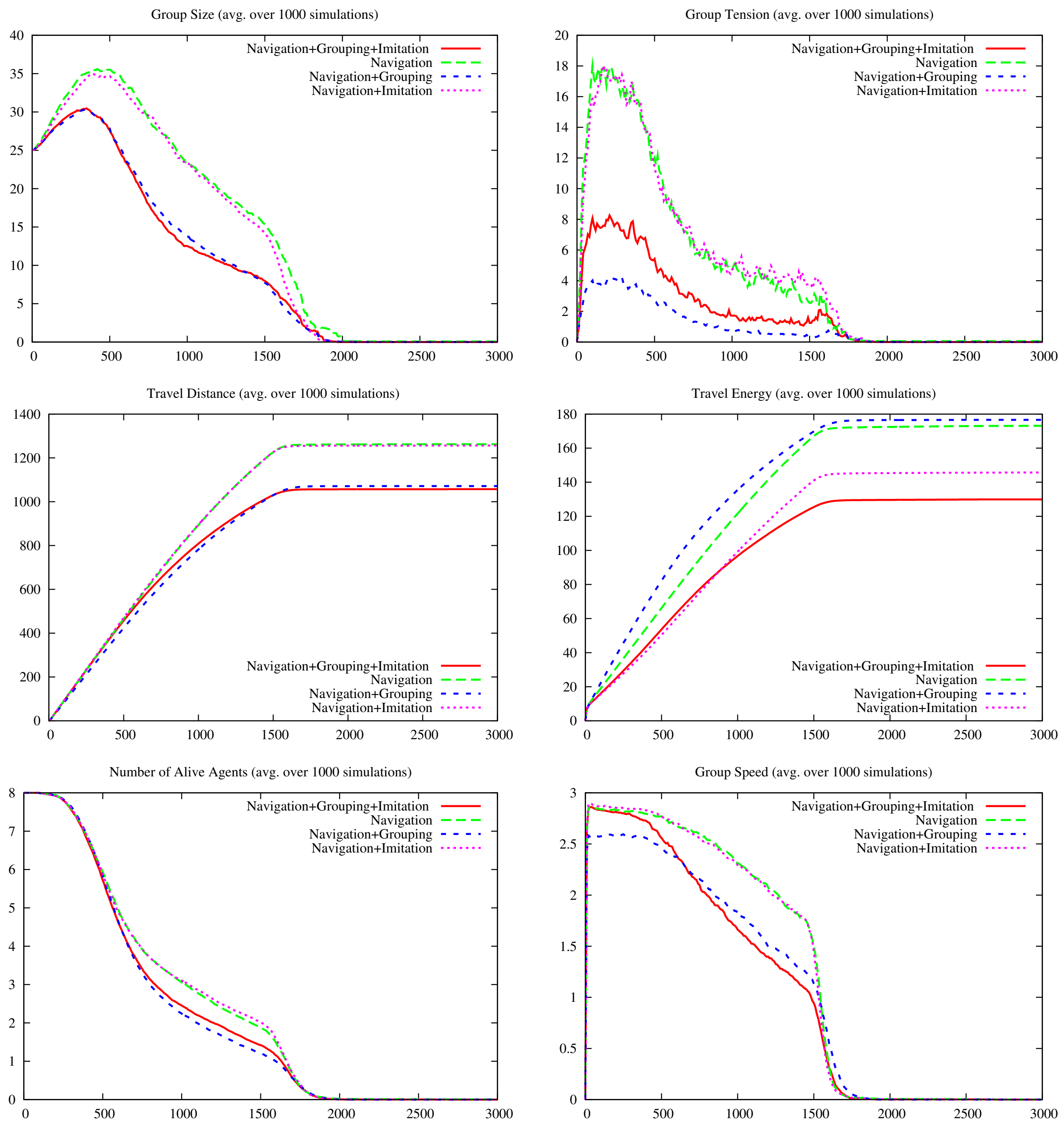
RL Navigation
Only Red team has weapons
Blue team: 8 agents navigating from source to destination
Red team: 1 agent attracted to Blue team

Figure C.1: Simulation set #1



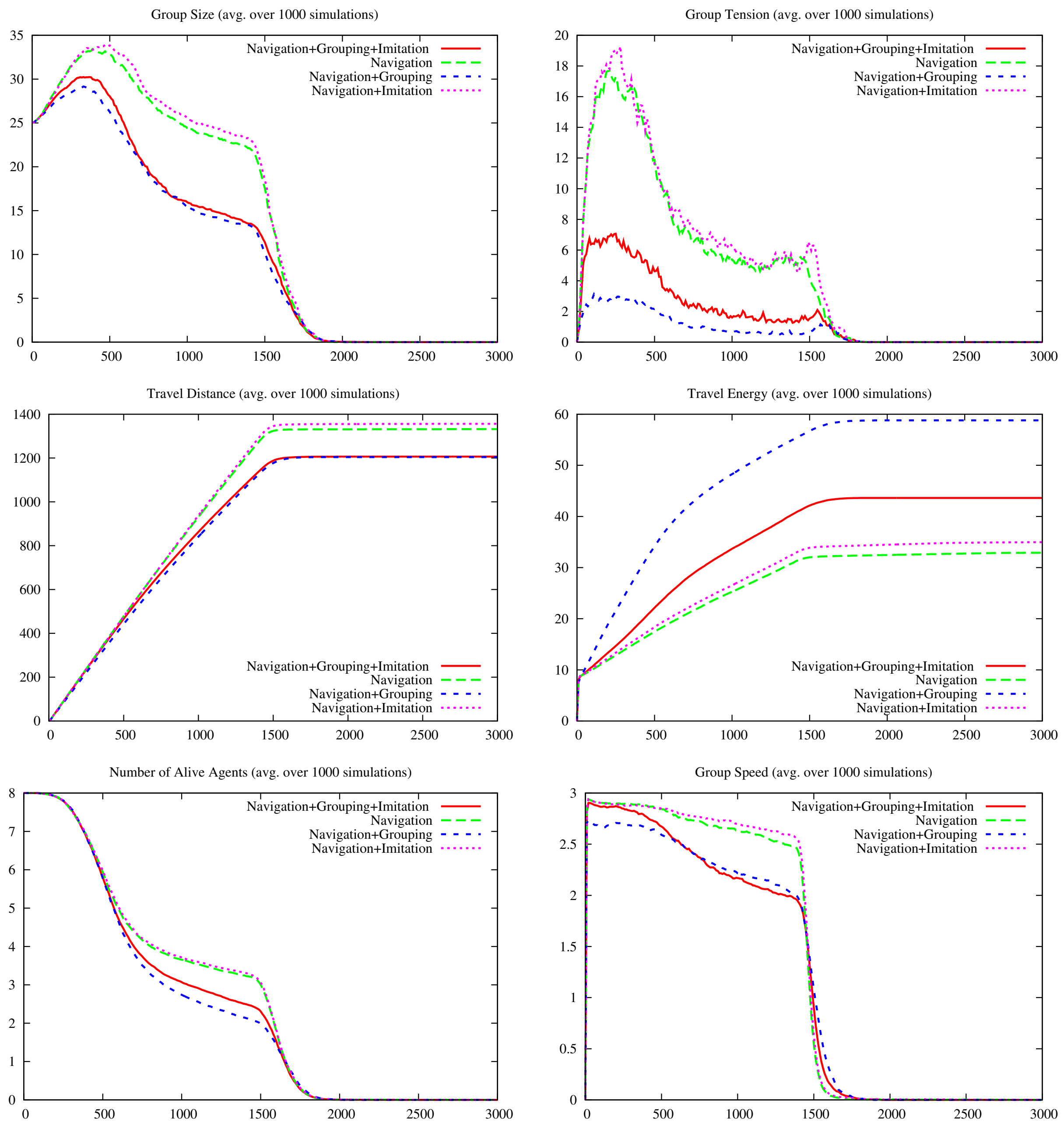
Gradient Navigation
Only Red team has weapons
Blue team: 8 agents navigating from source to destination
Red team: 1 agent attracted to Blue team

Figure C.2: Simulation set #2



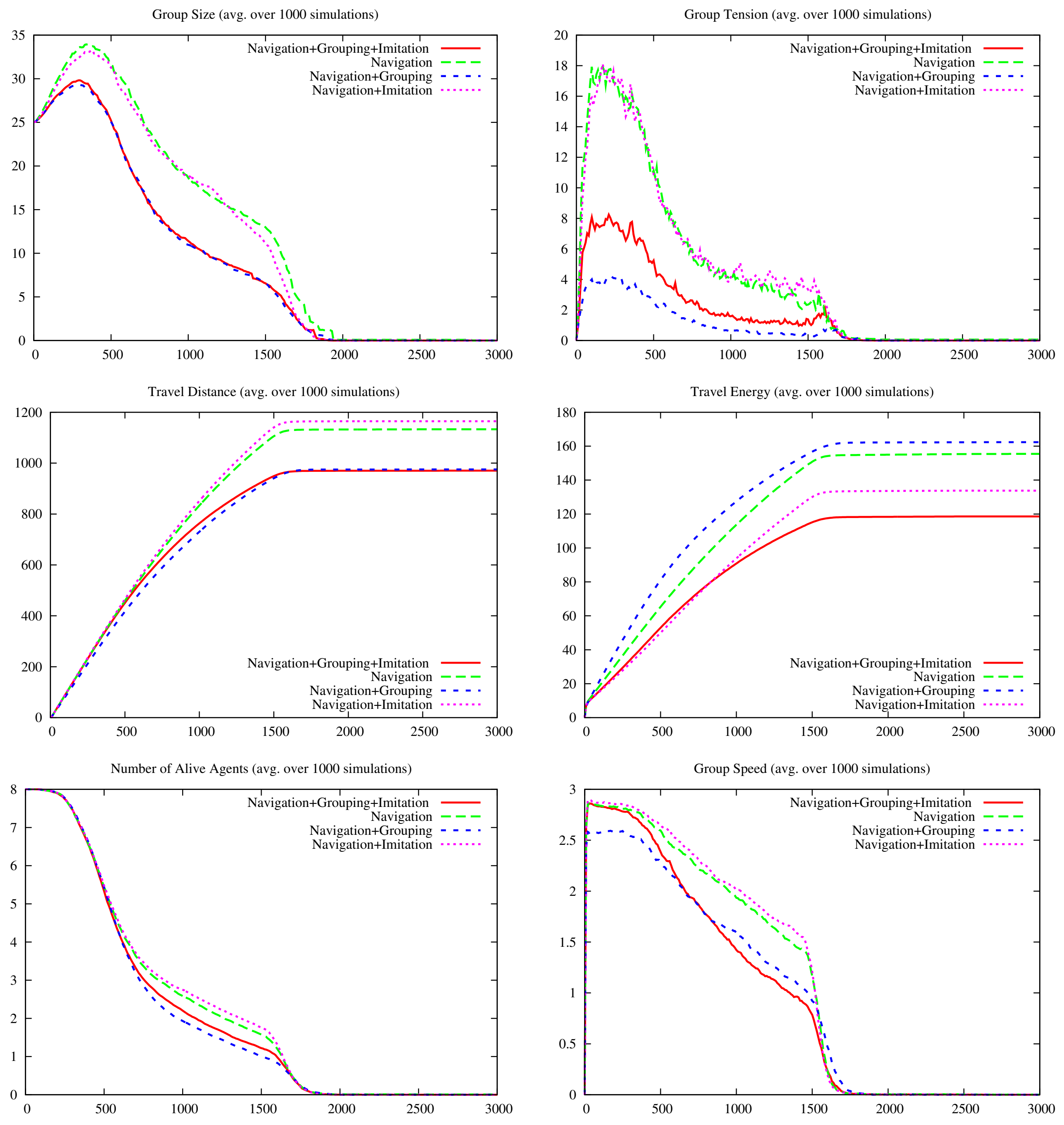
RL Navigation
Only Red team has weapons
Blue team: 8 agents navigating from source to destination
Red team: 3 agents attracted to Blue team

Figure C.3: Simulation set #3



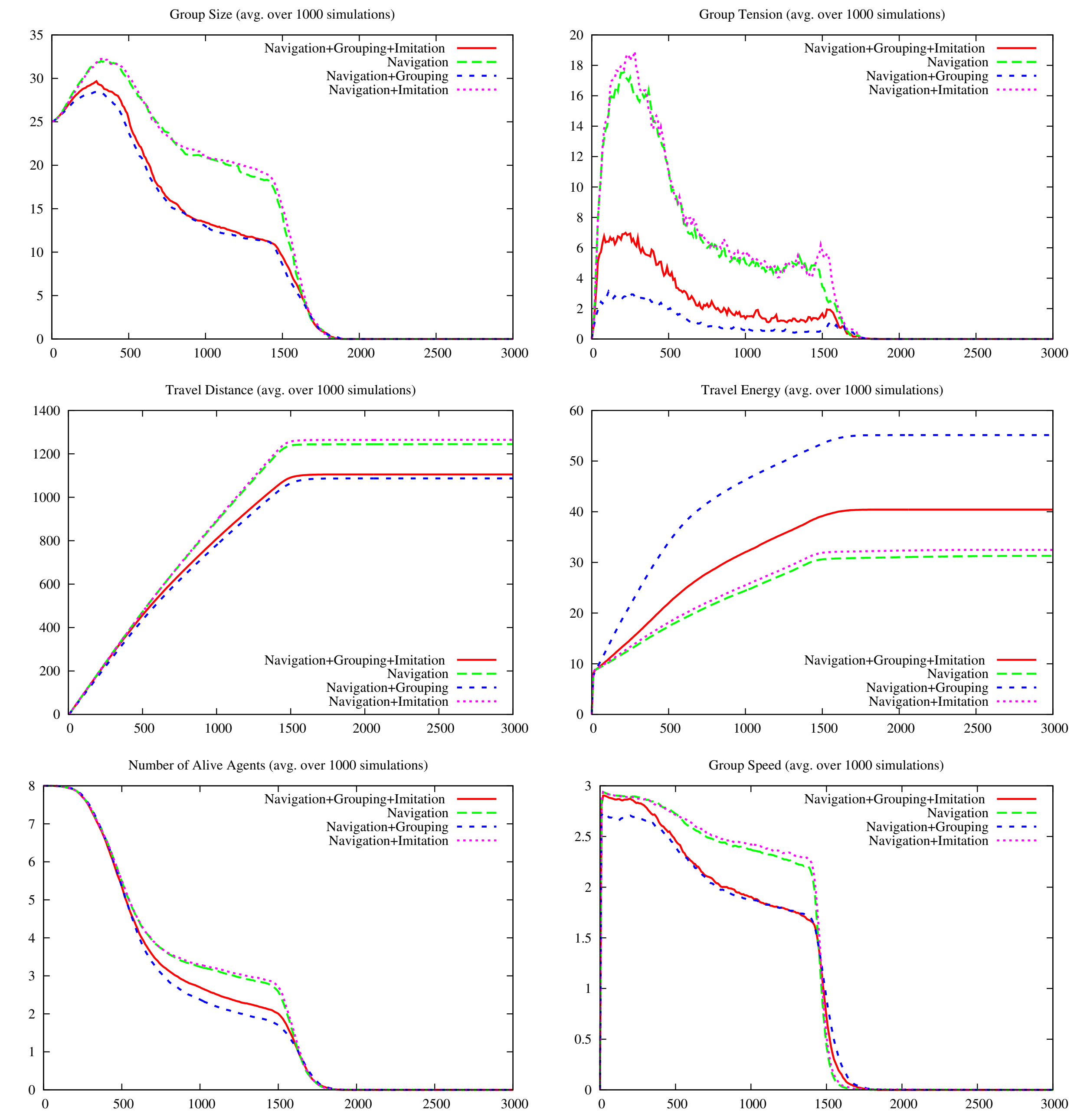
Gradient Navigation
Only Red team has weapons
Blue team: 8 agents navigating from source to destination
Red team: 3 agents attracted to Blue team

Figure C.4: Simulation set #4



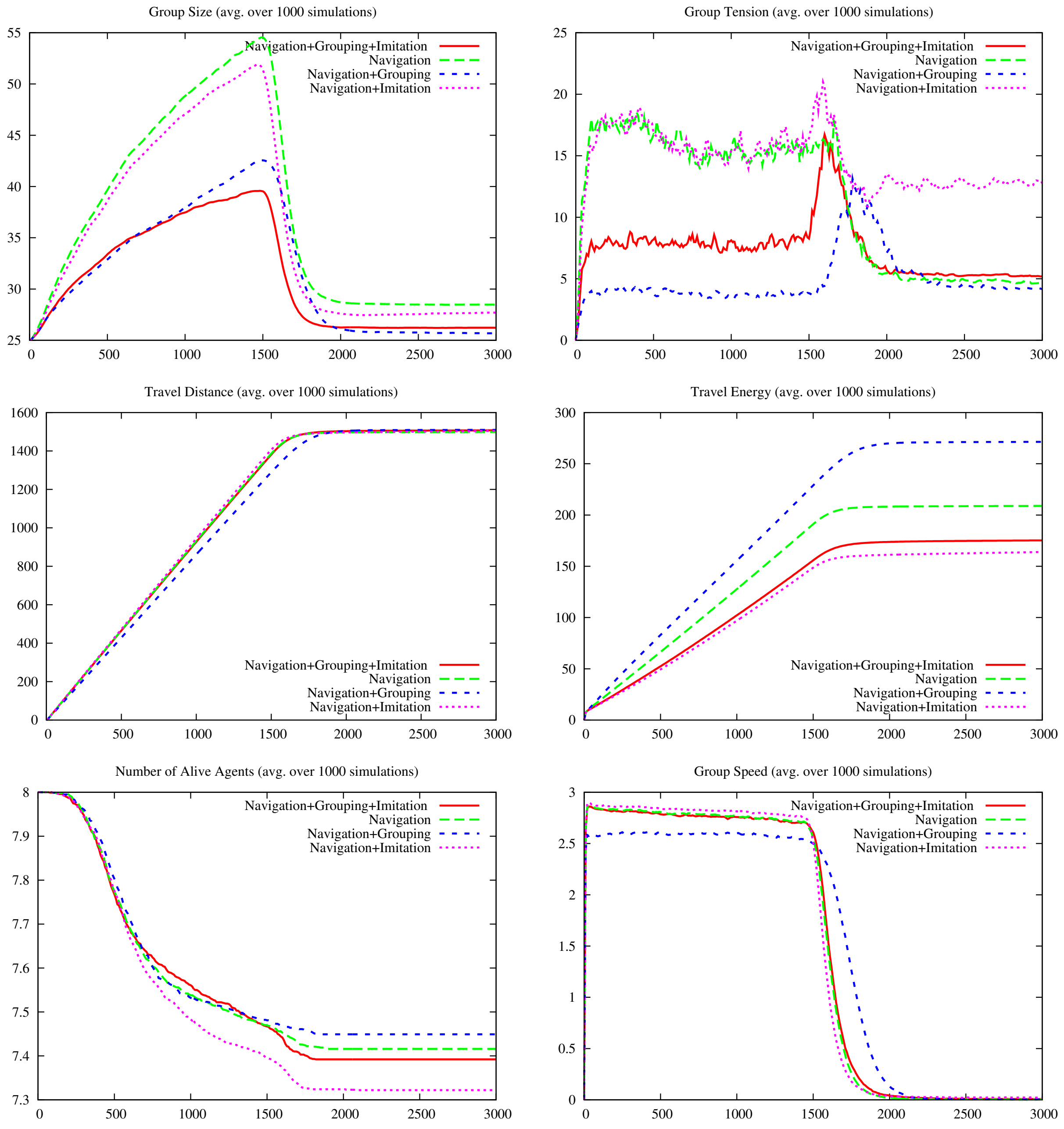
RL Navigation
Only Red team has weapons
Blue team: 8 agents navigating from source to destination
Red team: 5 agents attracted to Blue team

Figure C.5: Simulation set #5



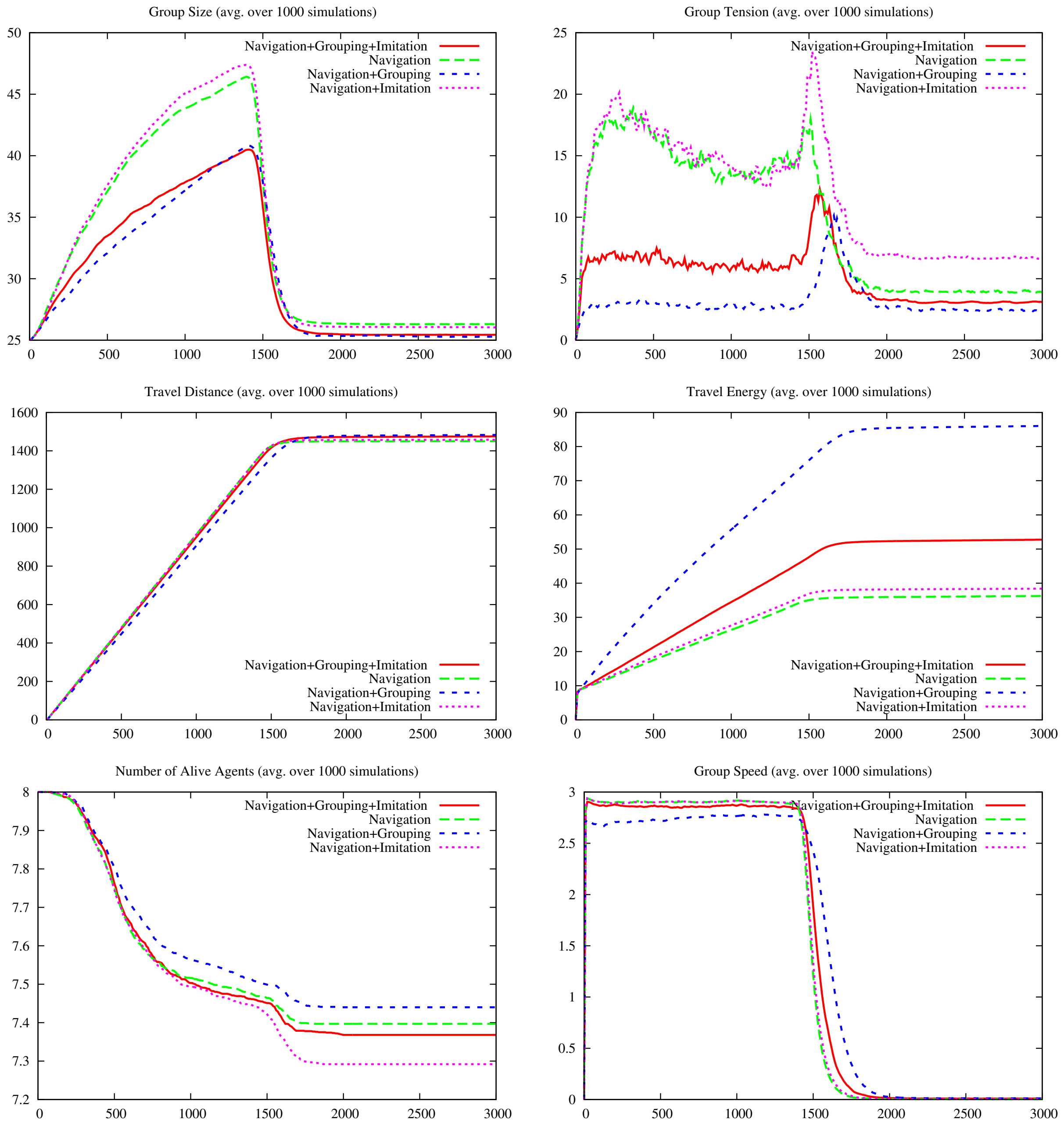
Gradient Navigation
Only Red team has weapons
Blue team: 8 agents navigating from source to destination
Red team: 5 agents attracted to Blue team

Figure C.6: Simulation set #6



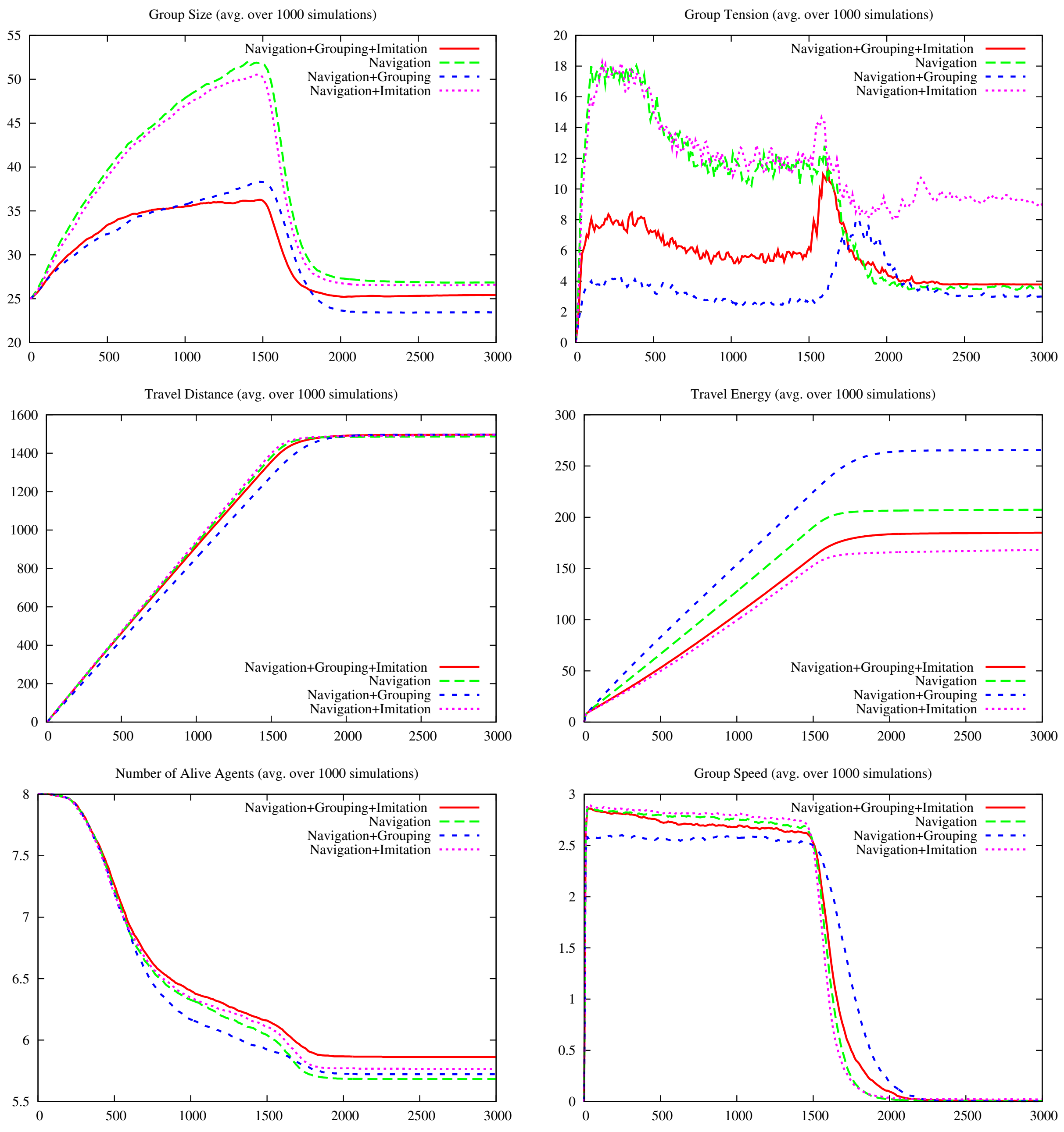
RL Navigation
Both teams have weapons
Blue team: 8 agents navigating from source to destination
Red team: 1 agent attracted to Blue team

Figure C.7: Simulation set #7



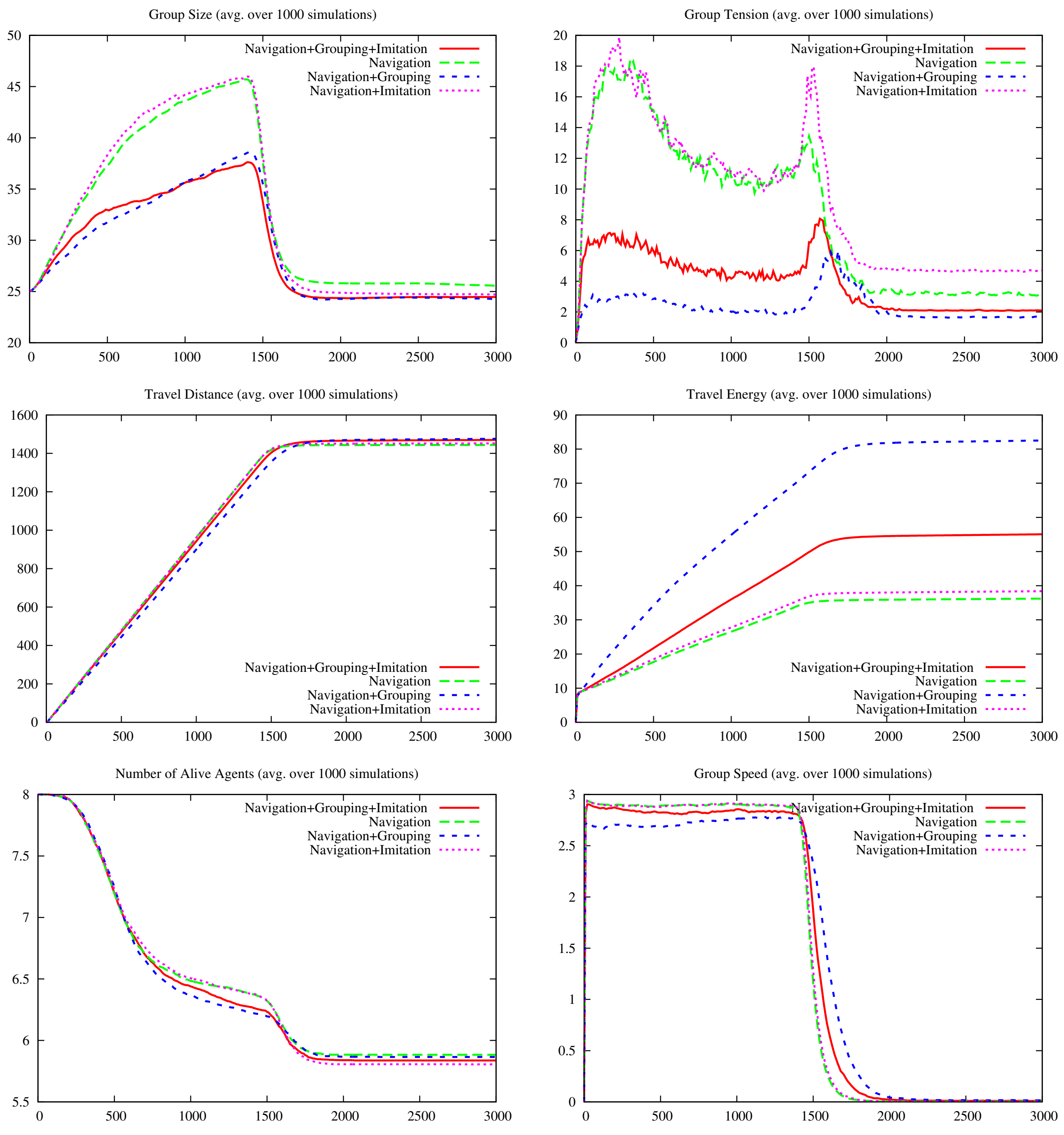
Gradient Navigation
Both teams have weapons
Blue team: 8 agents navigating from source to destination
Red team: 1 agent attracted to Blue team

Figure C.8: Simulation set #8



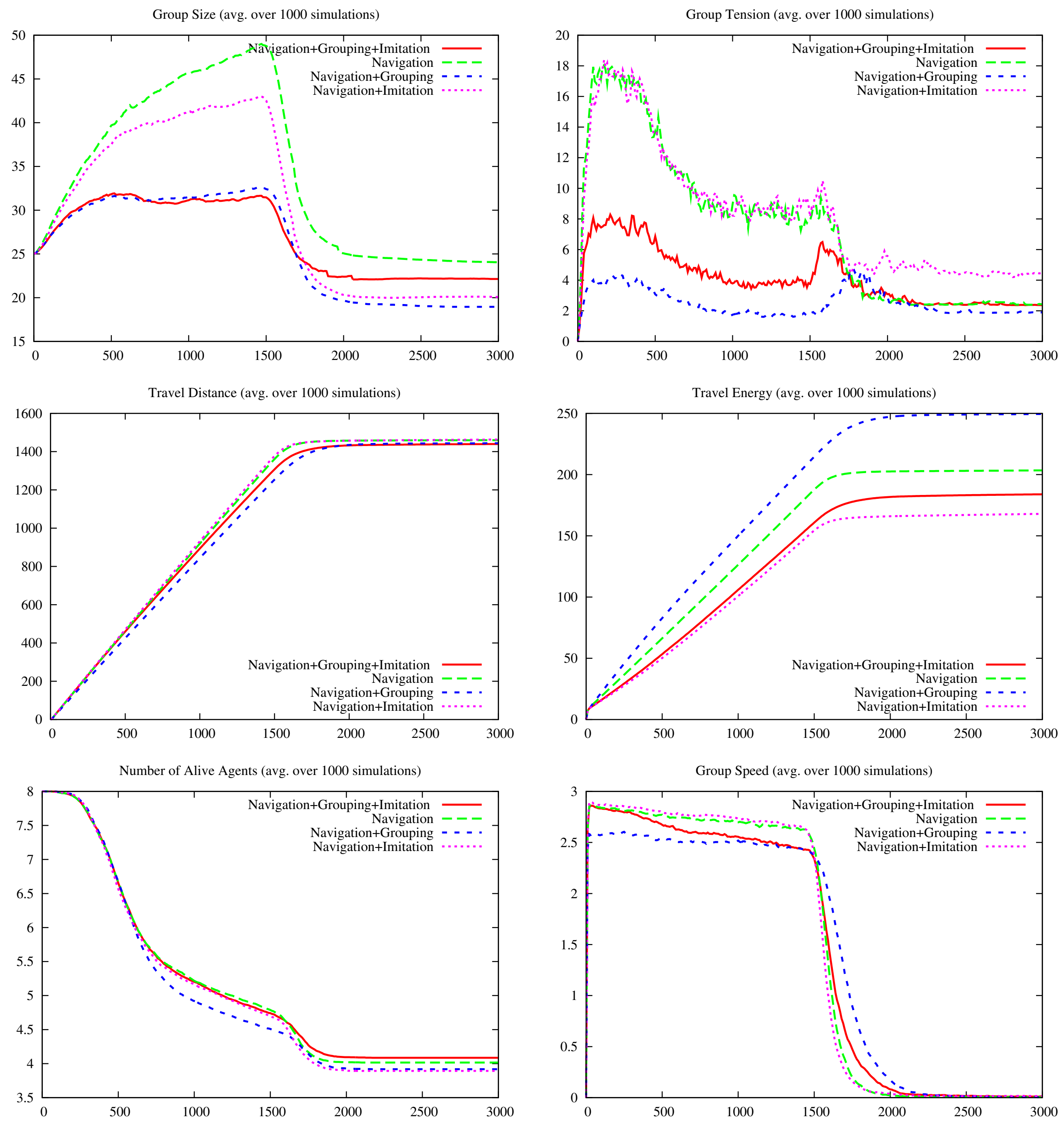
RL Navigation
Both teams have weapons
Blue team: 8 agents navigating from source to destination
Red team: 3 agents attracted to Blue team

Figure C.9: Simulation set #9



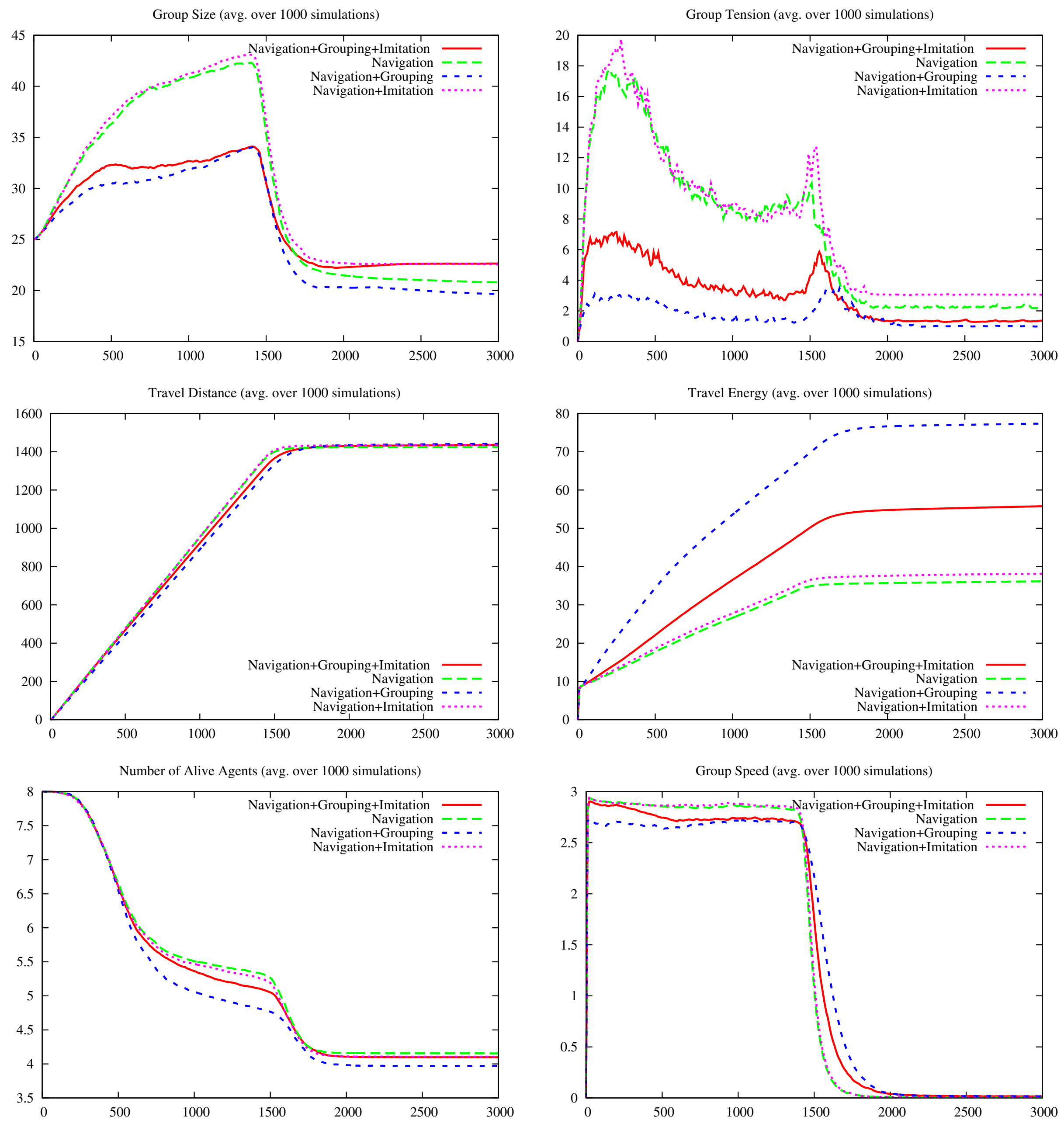
Gradient Navigation
Both teams have weapons
Blue team: 8 agents navigating from source to destination
Red team: 3 agents attracted to Blue team

Figure C.10: Simulation set #10



RL Navigation
Both teams have weapons
Blue team: 8 agents navigating from source to destination
Red team: 5 agents attracted to Blue team

Figure C.11: Simulation set #11



Gradient Navigation
Both teams have weapons
Blue team: 8 agents navigating from source to destination
Red team: 5 agents attracted to Blue team

Figure C.12: Simulation set #12

LIST OF REFERENCES

[Ark98] Ronald C. Arkin. *Behavior-Based Robotics*. The MIT Press, Cambridge, Massachusetts, 1998.

[Bal98] Tucker Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, 1998.

[BHA97] H. Badaroglu, U. Halici, I. Aybay, and C. Cerkez. “Digital Neuron Network Chip for the Random Neural Network Model with Programmable Architecture.” In *Proc. of ISCIS’97*, pp. 412–418, 1997.

[BK00] Hakan Bakırcıoğlu and Taşkın Koçak. “Survey of random neural network applications.” *European Journal of Operational Research*, **126**:319–330, 2000.

[BM93] Patricia T. Boyd and Stephen L. W. McMillan. “Chaotic scattering in the gravitational three-body problem.” *Chaos*, **3**(4):507–523, October 1993.

[BMD02] Loic Barthe, Benjamin Mora, Neil Dodgson, and Malcolm Sabin. “Triquadratic Reconstruction for Interactive Modelling of Potential Fields.” In *International Conference on Shape Modeling and Applications 2002 (SMI’02)*, p. 145, 2002.

[BN95] M. Bajura and U. Neumann. “Dynamic registration correction in video-base d reality systems.” *IEEE Computer Graphics and Applications*, **15**(5):52–60, 1995.

[Bro86] Rodney A. Brooks. “A robust layered control system for a mobile robot.” *IEEE Journal of Robotics and Automation*, **RA-2**(1):14–23, March 1986.

[Bro99] Rodney A. Brooks. *Cambrian Intelligence: The Early History of The New AI*. The MIT Press, Cambridge, Massachusetts, 1999.

[CG98] A.K. Cebrowski and J. J. Garstka. “Network centric warfare: its origin and future.” *Naval Institute Proceedings*, **124**(1):28–35, 1998.

[Dij59] Edsger Wybe Dijkstra. “A Note on Two Problems in Connexion with Graphs.” *Numerische Mathmatik*, **1**:269–271, 1959.

[FM98] Kikuo Fujimura and Mihail Makarov. “Foldover-free image warping.” *Graph. Models Image Process.*, **60**(2):100–111, 1998.

[Gel89] Erol Gelenbe. “Random neural networks with negative and positive signals and product form solution.” *Neural Computation*, **1**(4):502–511, 1989.

[Gel90] Erol Gelenbe. “Stability of the random neural network model.” *Neural Computation*, **2**(2):239–247, 1990.

[Gel93] Erol Gelenbe. “Learning in the recurrent random neural network.” *Neural Computation*, **5**(1):154–164, 1993.

[GHK05] Erol Gelenbe, Khaled Hussain, and Varol Kaptan. “Simulating autonomous agents in augmented reality.” *Journal of Systems and Software*, **74**(3):255–268, February 2005.

[GKH04] Erol Gelenbe, Varol Kaptan, and Khaled Hussain. “Simulating the navigation and control of autonomous agents.” In Per Svensson and Johan Schubert, editors, *Proceedings of the Seventh International Conference on Information Fusion*, volume I, pp. 183–189, Mountain View, CA, June 2004. International Society of Information Fusion.

[GKW04] Erol Gelenbe, Varol Kaptan, and Yu Wang. “Biological Metaphors for Agent Behavior.” In *Computer and Information Sciences - ISCIS 2004: 19th International Symposium*, volume 3280 of *Lecture Notes in Computer Science*, pp. 667–675. Springer-Verlag, October 2004.

[GKW05] Erol Gelenbe, Varol Kaptan, and Yu Wang. “Simulation and Modelling of Adversarial Games.” In *GAME-ON 2005*, Leicester, UK, November 2005.

[GKW06] Erol Gelenbe, Varol Kaptan, Yu Wang, Nick S. Walmsley, P. Gardiner, P.V. Pearce, and J. Moffat. “A Dynamic Model for Identifying Enemy Collective Behaviour.” In *11th ICCRTS - Coalition Command and Control in the Networked Era*, Cambridge, UK, September 2006.

[GM05] James Gain and Patrick Marais. “Warp Sculpting.” *IEEE Transactions on Visualization and Computer Graphics*, **11**(2):217–227, 2005.

[GML99] Erol Gelenbe, Zhi-Hong Mao, and Yan-Da Li. “Function approximation with spiked random networks.” *IEEE Transactions on Neural Networks*, **10**(1):3–9, January 1999.

[GP98] Erol Gelenbe and Guy Pujolle. *Introduction to Queueing Networks*. John Wiley & Sons, second edition, 1998.

[GSX01] Erol Gelenbe, Esin Şeref, and Zhiguang Xu. “Simulation with Learning Agents.” *Proceedings of IEEE*, **89**(2):148–157, February 2001.

[GW06] Erol Gelenbe and Yu Wang. “A mathematical approach for mission planning and rehearsal.” In Raja Suresh, editor, *Defense Transformation and Network-Centric Systems*, volume 6249 of *Proc. SPIE*, pp. 197–207, May 2006.

[GXS99] Erol Gelenbe, Zhiguang Xu, and Esin Şeref. “Cognitive Packet Networks.” In *IEEE Eleventh International Conference on Tools with Artificial Intelligence*, pp. 47–54, Chicago, Illinois, November 1999.

[Hal97] Ugur Halici. “Reinforcement Learning in Random Neural Networks for Cascaded Decisions.” *Journal of Biosystems, Elsevier*, **40**(1):83–91, January 1997.

[Hal99] Ugur Halici. “Reinforcement learning with internal expectation for the random neural network.” *European Journal of Operations Research*, 1999.

[HHT02] S. Hagan, S. R. Hameroff, and J. A. Tuszyński. “Quantum computation in brain microtubules: Decoherence and biological feasibility.” *Physical Review E*, **65**(6):061901, June 2002.

[HL01] David L. Hall and James Llinas, editors. *Handbook of Multisensor Data Fusion*. CRC Press, June 2001.

[IFF96] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. “Lua - an extensible extension language.” *Software: Practice & Experience*, **26**(6):635–652, 1996.

[KCP92] James R. Kent, Wayne E. Carlson, and Richard E. Parent. “Shape Transformation for Polyhedral Objects.” *Computer Graphics*, **26**(2):47–54, 1992.

[Ken99] Harold Kennedy. “Simulation Reshaping Military Training: technology jumping from teenagers’ computers to pilots’ cockpits.” *National Defense Magazine*, November 1999.

[KG06] Varol Kaptan and Erol Gelenbe. “Fusing terrain and goals: agent control in urban environments.” In Belur V. Dasarathy, editor, *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2006*, volume 6242 of *Proc. SPIE*, pp. 71–79, April 2006.

[Kha86] Oussama Khatib. “Real time obstacle avoidance for manipulators and mobile robots.” *International Journal of Robotics Research*, **5**(1):90–99, 1986.

[Koc97] Christof Koch. “Computation and the single neuron.” *Nature*, **385**:207–210, January 1997.

[LBR04] James Llinas, Christopher Bowman, Galina Rogowa, Alan Steinberg, Ed Waltz, and Frank White. “Revisiting the JDL Data Fusion Model II.” In *Proceedings of The 7th International Conference on Information Fusion*, Stockholm, Sweden, 2004.

[LP98] Shaun W. Lawson and J. R. Pretlove. “Augmented reality for underground pipe inspection and maintenance.” In *International Symposium on Intelligent Systems and Advanced Manufacturing, Telemanipulator and Telepresence Technologies V*, volume 3524 of *Proceedings of SPIE*, pp. 98–104, Boston, USA, November 1998.

[LS97] Aristidis Likas and Andreas Stafylopatis. “High Capacity Associative Memory Based on the Random Neural Network Model.” *International Journal of Pattern Recognition and Artificial Intelligence*, **10**(8):919–937, 1997.

[LS00] A. Likas and A. Stafylopatis. “Training the random neural network using quasi-Newton methods.” *European Journal of Operational Research*, **126**:331–339, 2000.

[Mae90] Patti Maes. “Situated Agents Can Have Goals.” In Patti Maes, editor, *Designing Autonomous Agents*, pp. 49–70. MIT Press, 1990.

[Mat97] Maja J. Mataric. “Reinforcement learning in the multi-robot domain.” *Autonomous Robots*, **4**(1):73–83, 1997.

[Mel95] J. P. Mellor. *“Enhanced reality visualization in a surgical environment.”*. Master’s thesis, MIT, Department of Electrical Engineering and Computer Science, January 1995.

[OF96] Norihiko Ono and Kenji Fukumoto. “Multi-agent reinforcement learning: A modular approach.” In *Proc. of Second International Conference on Multi-Agent Systems*, pp. 252–258, Kyoto, Japan, 1996. AAAI Press.

[OF97] Norihiko Ono and Kenji Fukumoto. “A modular approach to multi-agent reinforcement learning.” In Gerhard Weiss, editor, *Distributed Artificial Intelligence Meets Machine Learning*, volume 1221 of *Lecture Notes in Artificial Intelligence*, pp. 25–39. Springer-Verlag, 1997.

[opea] “The OpenFlight 3D database standard.” `http://www.openflight.org/products/standards/openflight/index.shtml`.

[opeb] “The OpenGL Graphics Standard.” `http://www.opengl.org`.

[Pan04] “Panel discussion on Challenges in higher level fusion: Unsolved, difficult, and misunderstood problems/approaches in levels 2-4 fusion research.” In Per Svensson and Johan Schubert, editors, *Proceedings of the Seventh International Conference on Information Fusion*, volume I, pp. 523–541, Mountain View, CA, Jun 2004. International Society of Information Fusion. Organized by Ivan Kadar, Moderated by Ivan Kadar and Per Svensson.

[Pen89] Roger Penrose. *The emperor’s new mind: concerning computers, minds, and the laws of physics*. Oxford University Press, Inc., New York, NY, USA, 1989.

[Pot99] Dave C. Pottinger. “Implementing Coordinated Movement.” *Game Developer Magazine*, pp. 48–58, January 1999.

[Rey87] Craig W. Reynolds. “Flocks, Herds, and Schools: A Distributed Behavioral Model.” *Computer Graphics*, **21**(4):25–34, 1987.

[Rey99] Craig W. Reynolds. “Steering Behaviors for Autonomous Characters.” In *Game Developer Conference*, pp. 763–782, San Jose, California, 1999.

[RT95] J. Rosenblatt and C. Thorpe. “Combining multiple goals in a behavior-based architecture.” In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, pp. 136–141, Pittsburgh, PA, August 1995.

[RW99] John H. Reif and Hongyan Wang. “Social Potential Fields: A Distributed Behavioral Control for Autonomous Robots.” *Robotics and Autonomous Systems*, **27**(3):171–194, March 1999.

[SB04] Alan Steinberg and Christopher Bowman. “Rethinking the JDL Data Fusion Levels.” In *NSSDF JHAPL*, June 2004.

[Sch04] Charles W. Schmidt. “The Price of Preparing for War.” *Environ Health Perspect.*, **112**(17):A1004–A1005, December 2004.

[SRK99] Alessandro Saffiotti, Enrique H. Ruspini, and Kurt Konolige. “Using Fuzzy Logic for Mobile Robot Control.” In H. J. Zimmermann, editor, *Practical Applications of Fuzzy Technologies*, volume 6 of *Handbook of Fuzzy Sets*, chapter 5, pp. 185–205. Kluwer Academic, MA, 1999.

[Ste93] Luc Steels. “The Artificial Life Roots of Artificial Intelligence.” *Artificial Life*, **1**:75–110, 1993.

[Str94] Steven H. Strogatz. *Nonlinear Dynamics and Chaos*. Perseus Books, Cambridge, MA, 1994.

[Tan93] Ming Tan. “Multi-Agent Reinforcement Learning: Independent versus Cooperative Agents.” In *Tenth International Conference on Machine Learning*, pp. 330–337, Amherst, Massachusetts, 1993.

[Teg00] Max Tegmark. “The importance of quantum decoherence in brain processes.” *Physical Review E*, **61**:4194, 2000.

[TM91] Demetri Terzopoulos and Dimitris Metaxas. “Dynamic 3D models with local and global deformations: deformable superquadrics.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**(7):703–714, 1991.

[Yap02] Peter Yap. “Grid-based Path-finding.” In *AI ’02: Proceedings of the 15th Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, pp. 44–55, London, UK, 2002. Springer-Verlag.

[YHM04] Han-Bing Yan, Shi-Min Hu, and Ralph Martin. “Morphing based on strain field interpolation.” *Computer Animation and Virtual Worlds*, **15**(3-4):443–452, 2004.