The Pennsylvania State University
The Graduate School

A GAME THEORETIC APPROACH TO MULTI-AGENT SYSTEMS IN HIGHLY DYNAMIC, INFORMATION-SPARSE, ROLE ASSIGNMENT SCENARIOS

A Thesis in
Computer Science and Engineering
by
Kyle Hollins Wray

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2012

*Signatures are on file in the Graduate School.

Abstract

The wide array of problems found in the field of Multi-Agent Systems (MAS) require rapid coordination toward a desired system state amidst the often limited and inaccurate information observed by the agents within. This thesis investigates the conditions surrounding this kind of emergent coordination. We focus primarily on a problem entitled Defender and provide a solution that leverages game theory in its agents' construction. Defender is a multi-predator multi-prey problem in which the predators seek to capture as many prey as possible before any of the prey reach their haven. The problem itself is a continuous noisy version of scenarios found throughout the MAS literature, particularly in the RoboCup, an annual robotic soccer competition. As such, it defines two sides as goal regions with its agents restricted to observational noise and a limited field of view. It specifically examines dynamically changing versions of learning algorithms like fictitious play, rational learning, and regret matching, in addition to baseline algorithms following minimax regret and greedy strategies. These algorithms operate in an anti-coordination game, which only rewards agents that select different actions from one another. Furthermore, this thesis presents optimizations to these algorithms and tests their robustness in ad hoc team scenarios. Results show that fictitious play outperforms all other algorithms and tends toward strong cooperative behavior.

In order to improve the algorithms in these kinds of problems, we also developed an adaptive communication architecture. The Distributed Communication Architecture (DCA) algorithm addresses the common issue found in real-world systems, specifically the message passing structure between agents. It is able to dynamically assign time slots to agents in a completely distributed manner. We apply this algorithm to Defender and show that while communication is not required to achieve emergent cooperation, it can improve the quality of agent's observations and subsequent decisions. Overall, this thesis investigates these primary topics to address the problem of constructing a team of agents seeking to cooperatively behave in a highly dynamic environment.

Table of Contents

List of Figures **vii**

List of Tables **x**

List of Symbols **xii**

Acknowledgments **xiv**

Chapter 1
Introduction **1**
1.1 Research Description 1
1.2 Overview of Thesis 2

Chapter 2
The State of the Art **4**
2.1 Foundations of Game Theory 4
2.1.1 Learning in Games 5
2.2 Multi-Agent Systems 6
2.2.1 An Overview of the Field 6
2.2.2 Predator-Prey and Pursuit-Evasion Problems 9
2.2.3 Multi-Agent Learning 11
2.2.4 Teammate Modeling 12
2.3 Communication Architectures 13
2.3.1 Computer Networking and Sensor Networks 13
2.3.2 Communications in Multi-Agent Systems 14
2.4 Summary 15

Chapter 3
Preliminary Material **16**
3.1 Multi-Agent Systems 16
3.1.1 Towards a Formal Definition 16
3.1.2 The Predator-Prey Problem 17
3.2 Game Theory 17
3.2.1 Normal Form and Solution Concepts 17

3.2.2 Learning Algorithms 20
3.2.3 Partial Observability and Uncertainty 22
3.3 Tracking 23
3.3.1 Extended Kalman Filter 23
3.3.2 Score and Threshold Gating Mechanisms 24
3.4 Communication 26
3.4.1 Channels and Broadcasting 26
3.4.2 Slot-Based Communication 27
3.5 Summary 27

Chapter 4
The Defender Problem **28**
4.1 Description 28
4.2 Formal Model 29
4.2.1 Agents 29
4.2.2 Environmental Parameters 29
4.2.3 The State Transition 30
4.3 The Game Structure 30
4.3.1 Formal Statement 31
4.3.2 An Example 32
4.4 The Tracker 35
4.4.1 Extended Kalman Filter and Gating 35
4.4.2 Action Estimation for Predators 36
4.4.3 System Behavior 36
4.5 Learning Algorithms 37
4.6 Computational Complexity and Optimizations 39
4.6.1 Complexity Analysis 40
4.6.2 Properties of Fictitious Play and Rational Learning 40
4.6.3 Quantity of Zeros 41
4.6.4 Optimized Algorithm 45
4.6.5 Optimization Experimentation 48
4.7 Experimentation 51
4.7.1 Overall Results 51
4.7.2 Selected Examples 55
4.7.3 Modifying Environmental Parameters 58
4.7.4 Ad Hoc Teams 60
4.8 Discussion 62
4.8.1 Emergent Commitment Behavior 62
4.8.2 Difficulties in Dynamic Games 64
4.8.3 Modifications to the Environment and Team Composition 65
4.8.4 Practical Implementations 66
4.9 Summary 66

Chapter 5
Ad Hoc Distributed Communication Architecture **68**
5.1 Description 68
5.2 Problem Statement 69
5.3 The Distributed Communication Architecture (DCA) 70

5.3.1 Base Functions 70
5.3.2 Learning, Slot Selection, and Settled States 71
5.3.2.1 The Learning State 72
5.3.2.2 The Slot Selection State 73
5.3.2.3 The Settled State 74
5.3.3 Slot Organization 74
5.4 Extensions 75
5.4.1 Inclusion of Local Agent Knowledge 75
5.4.2 Non-Sharing Variant 76
5.5 Application to Defender 77
5.6 Experimentation 78
5.6.1 Convergence Rates 78
5.6.2 Message Clarity Percentage 79
5.6.3 Examples 81
5.6.4 Defender Communication Experiments 81
5.7 Discussion 84
5.8 Summary 86

Chapter 6
Conclusion and Future Work **87**

Appendix A
Definitions and Theorems **89**
A.1 Probability Distributions 89

Appendix B
Predator Pseudocode **91**

Bibliography **97**

List of Figures

2.1 An example of the traditional predator-prey problem, as described by Benda et al. [7]. The blue squares represent the predators and the red square represents the prey. 9

3.1 An example of the gating problem encountered in tracking multiple agents. The blue square and grey cone represent an agent with a limited field-of-view. The green lines represent the two tracks' state observation histories. The red circle represents a new observation. 26
3.2 An example of slot-based communication in both time (x-axis) and frequency (y-axis) domains for a single system. 27

4.1 A visual example of the estimated path algorithm. The variables contained are found in algorithm 4.1. 32
4.2 A simplified example of three predators and three prey to explain Defender's payoffs using its normal form. Agent 1 observes no defenders and opponents $\{A, B, C\}$, agent 2 observes defender $\{1\}$ and opponents $\{A, B\}$, and agent 3 observes defenders $\{1, 2\}$ and opponents $\{A, B, C\}$. 34
4.3 A visual control flow diagram for Defender's learning algorithms. 39
4.4 A heat map demonstrating the percentage of zeros in a single predator's decision-making process. The red area is close to 100%, the yellow area is around 50%, and the blue area is around 0%. The x-axis and y-axis represent n and m, respectively. 44
4.5 The theoretical reduction in computation, visually elaborating on table 4.4. . . . 51
4.6 The percentage of simple prey captured. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8. 53
4.7 The percentage of maneuvering prey captured. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8. 53
4.8 The complete capture percentage against simple prey. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8. 54
4.9 The complete capture percentage against maneuvering prey. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8. . 54

4.10 This figure shows the average final location of the agents for each of the 5 algorithms. From left to right: fictitious play, rational learning, regret matching, minimax regret, and greedy. This describes four predators against four maneuvering prey. 55
4.11 An extended example of fictitious play predators chasing maneuvering prey ($n = m = 4$). The time sequence follows: top left, top right, middle left, middle right, bottom left, bottom right. Rational learning predators also show this kind of behavior. 56
4.12 An extended example of regret matching predators chasing maneuvering prey ($n = m = 4$). The time sequence follows: top left, top right, middle left, middle right, bottom left, bottom right. Minimax regret predators also show this kind of behavior. 57
4.13 An extended example of greedy predators chasing maneuvering prey ($n = m = 4$). The time sequence follows: top left, top right, middle left, middle right, bottom left, bottom right. 58
4.14 A summary of the effect that update rate has on the performance of four fictitious play predators chasing four straight moving prey. Each point represents the mean of 300 trials. 59
4.15 A visual representation of the overall results for ad hoc teams of fictitious play and rational learning predators from tables 4.8 and 4.9. Each data point on each line summarizes 1000 trials of that particular configuration. 61
4.16 A visual representation of the individual results for ad hoc teams of fictitious play predators from table 4.8. Each data point on each line summarizes 250 trials of that particular configuration. 61
4.17 A visual representation of the individual results for ad hoc teams of rational learning predators from table 4.9. Each data point on each line summarizes 250 trials of that particular configuration. 62

5.1 The high-level state transition diagram for the communication algorithm (top). A visual example of the communication cycle structure (bottom) with numeric values showing the settled slot ID's, "N" representing a noisy signal, and "-" denoting a silent signal. 71
5.2 A visual example of the experiment for four agents, for which each agent can observe other agents strictly above it. The number of agents visible is shown as the number on each agent. 76
5.3 A visual of the convergence rates (in time slots) for 1 to 50 agents with a simultaneous start. The general algorithm's rates are given on the left figure, and the algorithm with agent knowledge is given on in right figure. 79
5.4 A visual of the convergence rates (in time slots) for 1 to 50 agents with a random offset. The general algorithm's rates are given on the left figure, and the algorithm with agent knowledge is given on in right figure. 79
5.5 A visual of the proportion of clear messages (a percentage) for 1 to 50 agents with a simultaneous start. The general algorithm's proportions are given on the left figure, and the algorithm with agent knowledge is given on in right figure. 80
5.6 A visual of the proportion of clear messages (a percentage) for 1 to 50 agents with a random offset. The general algorithm's proportions are given on the left figure, and the algorithm with agent knowledge is given on in right figure. 80

5.7 Two examples of the general algorithm in a worst-case scenario (left) and staggered scenario (right). Each example consists of 10 agents in the system. The axes denote the time step and slot, with each line representing the agent's slot choice history. 81
5.8 Two examples of the variant algorithm with knowledge included in a worst-case scenario (left) and staggered scenario (right). Each example consists of 10 agents in the system. The axes denote the time step and slot, with each line representing the agent's slot choice history. 81
5.9 The percentage of simple prey captured. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6. 82
5.10 The percentage of maneuvering prey captured. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6. . . 83
5.11 The complete capture percentage against simple prey. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6. 83
5.12 The complete capture percentage against maneuvering prey. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6. 84

List of Tables

4.1 The normal form of the game depicted in figure 4.2. 34

4.2 The tabular example for the computation in $p_i^*(a_i)$ with $n = m = 3$. Each cell corresponds to a permutation and each letter in that permutation corresponds to the selected prey A, B, or C of observed predators 0, 1, and 2. From equation 4.10, we examine the case in which the $argmax$'s parameter a_i is A, and denote a *Yes* if an observed predator in a permutation cell also selects A. 42

4.3 Results for the computation time of the general algorithm (above), the optimized algorithm (middle), and the reduction in computation ratio (below) The number of predators n varies over the rows, and the number of prey m varies over the columns. Each cell represents 100 trials. 49

4.4 The ideal reduction in computation from the theoretical analysis of the number of zeros. The number of predators n varies over the rows, and the number of prey m varies over the columns. Each cell was computed by taking the maximum of the ratio of zero or one multiplications to total multiplications, e.g. for the zero case, $\frac{m^n-(m-1)^n}{m^n} = 1 - (\frac{(m-1)}{m})^n$. 50

4.5 Results for the control loop operation count of the general algorithm (above), the optimized algorithm (middle), and the reduction in computation ratio (below). The number of predators n varies over the rows, and the number of prey m varies over the columns. Each cell represents 100 trials. 50

4.6 Results for the percentage of prey captured in configurations of {simple, maneuvering} prey vs. {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 8 agents. These results summarize 5000 trials for each cell. 52

4.7 Complete capture percentage results for configurations of {simple, maneuvering} prey vs. {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 8 agents. These results summarize 5000 trials for each cell. 52

4.8 Results from ad hoc teams with fictitious play predators and one perturbed predator given by the row. Team sizes are given by each column. Each cell represents the mean over 250 trials. The final row states the cumulative results from that column, a total of 1000 trials per cell. 60

4.9 Results from ad hoc teams with rational learning predators and one perturbed predator given by the row. Team sizes are given by each column. Each cell represents the mean over 250 trials. The final row states the cumulative results from that column, a total of 1000 trials per cell. 60

5.1 Given each type of experiment and the number of agents, the convergence rates (in number of steps) are shown. The (K) signifies that local agent knowledge was included. 79
5.2 For each of the four types of experiments and number of agents, this table shows the average percentages of clear messages sent for a cycle. The (K) signifies that local agent knowledge was included. 80
5.3 Results for the percentage of prey captured in configurations of {simple, maneuvering} prey vs. *communicating* {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 6 agents. These results summarize 1000 trials for each cell. 82
5.4 Complete capture percentage results for configurations of {simple, maneuvering} prey vs. *communicating* {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 6 agents. These results summarize 1000 trials for each cell. 82

List of Symbols

A, a	Capital letters, like A, in a mathematical formula will denote a set unless otherwise stated. Lowercase letters, like a, denote elements within a set, e.g. $a \in A$. Set theoretic notation also applies: intersection $A \bigcap B$, union $A \bigcup B$, set minus $A \setminus B$, set size $\|A\|$, *et cetera.*
$\mathbb{R}$	The set of real numbers.
$\mathbb{N}$	The set of integers.
$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$	A vector $\vec{x}$, defined as a column of elements, each denoted as x_i
$\\|\vec{x}\\|_p$	The p-norm of the vector $\vec{x} \in \mathbb{R}^n$. Assume $p = 2$ unless otherwise stated. Formally, the 1-norm and 2-norm are defined as follows. $\\|\vec{x}\\|_1 = \|x_1\| + \ldots + \|x_n\|$ $\\|\vec{x}\\|_2 = \sqrt{\|x_1\|^2 + \ldots + \|x_n\|^2}$
$\vec{e} = \frac{1}{\sqrt{n}} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$	A normalized n-element vector that corresponds to the central point on the standard simplex in $\mathbb{R}^n$
$\mathcal{I}\{expression\}$	An indicator function that returns 0 or 1 based on the expression, i.e. $\mathcal{I}\{expression\} = \begin{cases} 1, & \text{if expression} = \textbf{true} \\ 0, & \text{otherwise} \end{cases}$
$O(f)$	The worst-case computational complexity, dominated by the function f.
$\mathcal{P}(X)$	For some set X, $\mathcal{P}(X)$ is the power set of X.
$p(x)$	A probability distribution at x.

$X \sim \mathcal{U}(a, b)$	The random variable X follows a Uniform distribution on the interval (a, b).
$X \sim \mathcal{N}(\mu, \sigma^2)$	The random variable X follows a Normal distribution with mean μ and variance σ^2.
$\min\{X\}$ or $\max\{X\}$	A function that returns the minimum/maximum of the set X with $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}$.
$\arg\min\{X\}$ or $\arg\max\{X\}$	A function that returns the index of the minimum/maximum value from the set X with $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}$.
$clamp(x, a, b)$	A clamp function that takes value x on the interval (a, b) and returns the bounds of the interval otherwise, i.e. $clamp(x, a, b) = \begin{cases} a, & \text{if } x < a \\ b, & \text{if } x > b \\ x, & \text{otherwise} \end{cases}$
$resolveAction(n, m, i, j)$	This function returns the action for a matrix representation of a normal form game given n players, m actions, cell i (matrix row), and a player j (matrix column). Assume zero-indexed definitions, i.e. $i \in \{0, \ldots, m^n - 1\}$ and $j \in \{0, \ldots, n - 1\}$. The function is defined as follows: $resolveAction(n, m, i, j) = \left\lfloor \frac{i}{m^j} \right\rfloor \mod m$
$agent(r)$	Return the agent $i \in N$ who performed the role $r \in R_i$. This function enables concise mathematical statements.

Acknowledgments

I would like to personally thank Dr. Russell Burkhardt and Dr. Benjamin Thompson for providing me with an endless supply of support and advice throughout my studies. This thesis would simply not have been possible without them. I also owe a debt of gratitude to the Pennsylvania State University's Applied Research Laboratory for their amazing program that supports the continued education of many individuals, of which I am humbled to call myself one of them. I deeply appreciate Dr. Vijaykrishnan Narayanan's support throughout my graduate education and for providing me with a home on campus in the Microsystems Design Laboratory. I also would like to thank Dr. Robert Collins for his excellent courses and for serving on my thesis committee.

This thesis would have not happened without the support of my friends and family. First, I would like to thank God for always being there to guide me throughout these difficult but rewarding years. My grandparents Edward Neuhauser and Barbra Neuhauser, who supported my initial investigations into programming and love of computation. My aunt and uncle Patty Wray and Doug Smith, who nurtured my curiosity about computers and technology from my earliest days to today. Lastly, my parents Peter Wray and Robin Wray, who have always encouraged me to strive for my dreams and supported me along the way.

This material is based upon work supported by The Office of Naval Research through The Naval Sea Systems Command under Contract No. N00024-02-D-6604. Their support is greatly appreciated.

Dedication

To my friends and family.

Chapter 1

Introduction

Current Multi-Agent Systems (MAS) research has moved toward a strong focus on uncertainty and dynamism for groups of distributed agents. This is evident by the number of robotic applications [47, 78, 90], the advent of ad hoc teams [81, 5], the development of role assignment algorithms [25, 14, 79], and in the design of communication among agents [29, 77, 24]. This thesis aligns itself with this important focus of the field. We explore the conditions for emergent cooperative behavior, leveraging the foundations of game theoretic learning, and expand upon these ideas through multiple means for inter-agent communication. To test our hypotheses, we consistently refer to a dynamic variant of the predator-prey problem [7] and clearly demonstrate that emergent cooperation can arise in these highly dynamic scenarios.

1.1 Research Description

In essence, multi-agent systems strive to fully describe a collection of agents, i.e. intelligent entities with some degree of autonomy, as they interact within an environment. The predator-prey problem has been a prominent forerunner for which multi-agent algorithms may be tested. It consists of a group of predators whose goal is to capture a prey moving randomly in a toroidal grid-world. Numerous variants exist that each underscore different components of the overarching coordination problem [5, 16, 85]. Our particular variant we call "Defender" describes a highly dynamic coordination situation for the predators, emulating football, rugby, or soccer in its design. The goal of the predators is to apprehend one of the multiple prey who move randomly towards the predators' goal region. Our problem models many of the common issues resulting from noise and lack of information found in realistic robotic applications.

We use this problem to investigate the broader question of agent coordination in information-sparse role-assignment scenarios. In particular, we examine the game theoretic learning algorithms: fictitious play, rational (Bayesian) learning, and regret matching, in addition to two baseline algorithms: minimax regret and a simple greedy strategy. Game theory provides a solid mathematical foundation in which to implement these "optimizing" decision-making algorithms, and is by far the most common approach to describe MAS. Traditionally, these learning algorithms center on well-posed problems such as war of attrition, stag hunt, or Shapley's game. We instead design an anti-coordination-like game in which a large proportion of information is unknown. In fact, the game itself changes slightly over each iteration, introducing numerous other issues such as how to handle old observations or what to do with newly added or removed players. We address these issues, enhance the computational performance of these algorithms, and present our findings when they are applied to Defender.

One way to alleviate the uncertainty problem in MAS is to provide high levels of communication. With the basic learning framework in place, we examine a direct method for inter-agent communication. The ad hoc distributed communication architecture enables agents to dynamically join, share, and leave a common round-robin communication system. Perhaps most importantly, it is able to form and be maintained without the need of any centralized computation. The application of this mechanism to Defender enables our predators to share their prey selections which reduces the overall uncertainty in the system.

In summary, this thesis contributes the following to the multi-agent systems community and its related fields:

- An investigation into the effects that very chaotic games have on the popular game theoretic learning algorithms: fictitious play, rational learning, and regret matching.
- The Defender Problem, a variant of the traditional predator-prey problem, enables future research on highly dynamic scenarios to have a common platform for which they may develop their algorithms.
- An optimized version of fictitious play and rational learning algorithms in anti-coordination games, often improving performance orders of magnitude over a direct implementation.
- Distributed Communication Architecture (DCA), an algorithm that creates a pattern of communication over dynamically changing time slots, managed entirely in a decentralized manner.
- A thorough literature review on the state of multi-agent systems, predator-prey, and agent communication algorithms.

1.2 Overview of Thesis

Chapter 2 provides a complete literature review on the topics related to realistic MAS. Namely, we cover the foundations of both multi-agent systems and game theory, in addition to the places in which they intersect such as learning algorithms. We also state previous research on teammate modeling and communication as it relates to our communication approaches.

Chapter 3 introduces the foundational concepts for game theory, multi-agent learning, and basic communication methods. Our goal for this chapter is to only provide the essential information needed for comprehension of the proceeding chapters.

Chapter 4 describes the Defender problem, our solution of an anti-coordination game with payoffs proportional to estimated travel distance, and game theoretic learning algorithms. We also investigate an interesting optimization for these algorithms based on the structure of anti-coordination games. Finally, this chapter provides detailed results from our simulation environment. The simulation supports our claim that our algorithm is a viable solution to Defender while simultaneously comparing the various learning algorithms to demonstrate action-based fictitious play's superiority over the others.

Chapter 5 continues our investigation into highly dynamic MAS. In it we provide our distributed communication architecture and a rigorous series of simulations to test its viability. We also attempt to improve upon it by introducing a variant for these Defender-like problems that incorporates local agent knowledge and as a result vastly improves the convergence rates.

Chapter 6 concludes with a summary of our work, stating the major results. We close with a discussion of future work in each of the topics. Two appendices and the references that were cited follow.

Chapter 2

The State of the Art

To put the contributions of our work into perspective, this chapter provides a thorough review of the current state of the art. We start by reviewing a select portion of the game theory literature, namely learning in games. Next we detail the general literature on multi-agent systems since it forms the foundations for the work presented here. After getting a broad sense of the field, we will cover the work done on predator-prey and pursuit-evasion problems, as Defender itself is included in this domain. Then we briefly describe some of the current research specific to multi-agent learning, agent inference, and teammate modeling methods. After this, we will briefly delve into the related work in the field of communication architectures and sensor networks, including those which intersect with the multi-agent community. This chapter concludes with a summary of the overall trends that we observed in the literature.

2.1 Foundations of Game Theory

The foundations of modern day *game theory* can be found in the now famous book written by Von Neumann and Morgenstern [91]. The field strives to fully describe the optimal behavior behind decisions made by rational players in games. Using this framework, one can describe any situation in which multiple entities are trying to obtain the best utility value for themselves by considering the actions of other players and the effect they will have on this value. One of the primary tools used in describing "optimal" behavior is known as Nash equilibrium [59, 60]. The essential idea of Nash equilibrium *strategy profile* is that all players are playing their *best response* to others. This means that no player can do better by selecting another action, given all others perform their particular actions as described by this strategy profile. This profound idea and method of constructing the games may be expanded in a multitude of directions. Our work focuses on repeated games of incomplete and imperfect information that also change in a structured manner over time. In summary, these are games in which the players do not necessarily know the number of other players playing or the number of actions available to each player. Several excellent books have been written on the subject [73, 63, 22]. Chapter 3 will detail the essential material of the field for the purposes of this thesis, e.g. the formulation of games and Nash equilibrium which is common throughout all of these books.

2.1.1 Learning in Games

In this thesis, we cover four main learning algorithms developed primarily from the field of game theory. The objective of these algorithms is to perform the optimal sequence of actions over time such that eventually the decisions optimize the utility gained at each stage. One of the earliest algorithms was developed by Brown [13] and Robinson [70], called *fictitious play.* It assumes that other players are playing a stationary mixed strategy, that may then be represented as a distribution over their corresponding actions. This allows for an expected utility to be generated, with the best action at each stage (there are various mathematical definitions for "best") following this value as it converges over iteration. Moreover, fictitious play is proven to converge to the Nash equilibrium of the game, under a few assumptions about the other players and the game itself. For example, one assumption states that the game must be either a two player game, a type of coordination game, or solvable via iterated dominance for pure-strategy solutions. It is these heavy assumptions that have brought about a number of adjustments over the years since its inception. A major issue with fictitious play lies in its unproven convergence for more than two players. Work done by Fudenberg and Kreps [19] attempted to generalize fictitious play, alleviating this predominant concern in addition to providing a solid formal understanding of its convergence properties.

Fictitious play has enjoyed a host of variants over the years, each one improving upon the last. Perhaps one of the most well-known is called *smooth fictitious play*, proposed by Fudenberg and Levine [21]. They showed that by adding a concave decreasing function that diminishes over time to fictitious play's computation, the players satisfy their universal conditional consistency condition. This condition allows for a much stronger proof of convergence than had been previously shown. The method was built off of previous work on *cautious fictitious play*, a modification that forces the original probability calculation to be adjusted exponentially based on the frequency as a function of its observations made over the history of stages [20]. Cautious fictitious play creates an emergent behavior that induces frequently played strategies to become probabilistically more inclined to be played again. The least played strategies will therefore be relatively ignored. A third adjustment entitled *dynamic fictitious play* assumes that the actions of other players may be modeled as a dynamic control system [72]. For two player games, this entails taking the gradients over the tracked history of player's actions in a continuous-time domain.

There are approaches to learning other than fictitious play that guarantee convergence to Nash or correlated equilibrium. One popular algorithm uses Bayesian updating to model the other player's actions' probability distributions. This method proposed by Kalai and Lehrer [42] is called *rational learning* (or Bayesian learning), and is shown to converge to Nash equilibrium. While rational learning follows the similar probabilistic modeling methodology as fictitious play, there is one other algorithm created by Hart and Mas-Colell [34] called *regret matching* that answers the question of how to optimally learn in repeated games much differently. In regret matching, players seek to minimize their regret over time, for which regret is defined as the difference between the average reward earned and the average reward the player could have obtained. It generates a ratio that may be used to weigh the player's decision among the available actions. The algorithm is shown to exhibit *no regret*, i.e. the difference previously described tends to zero as time goes to infinity.

Game theory provides many viable solutions to the multi-agent community's problems. In the next section, we will discuss some of the uses of game theory in multi-agent learning algorithms.

2.2 Multi-Agent Systems

We seek to understand game theoretic models applied to a limited information predator-prey domain. To this end, we first review related work on multi-agent systems, predator-prey (pursuit-evasion) problems, and multi-agent learning.

2.2.1 An Overview of the Field

Multi-Agent Systems (MAS) consist of a number of agents that are attempting to meet a global objective usually shared among a set of these agents. They often operate in an environment with limited communication and/or observation capabilities, making the global objectives difficult to achieve. The field originally grew out of Distributed Artificial Intelligence (DAI), and has now become the standard of describing both physical and virtual interacting entities. Initial work done by Wooldridge and Jennings [96], Lesser [50], and Stone and Veloso [80] laid the groundwork for the field by formulating the formal requirements of a MAS, identified the important issues therein, categorized the particular kinds of problems, and provided a direction for future research. Specifically, Wooldridge and Jennings investigated the definition of *agency* and the boundaries of such an *agent theory*. Lesser [50] gave an account of the shift from DAI to MAS and the major problems faced by the field. Stone and Veloso [80] divided the MAS into four main categories, based primarily on the heterogeneity of agents and their limitations on observation and communication. More recently, Horling and Lesser [35] sought to categorically describe the relational paradigms among agents. Panait and Luke [64] survey the contributions made in the pursuit of cooperative multi-agent learning, which has become one of the most popular topics in MAS. Lastly, Shoham and Keyton-Brown [73] have consolidated the main ideas surrounding game theory-based MAS into a concise volume for present and future researchers in the field.

Possibly the most well-known "proving ground" for multi-agent algorithms is called the RoboCup [47]. It is an annual robotic soccer competition in which two teams of robots must simply play the game of soccer. There is both a robotic competition and a virtual competition utilizing the Soccer Server, created by Noda et al. [62]. Stone and Veloso [79] and Stone [78] used reinforcement learning in their robotic teams while layering the tasks required for various maneuvers and set-plays. They also provided a formal definition for this domain: Periodic Time Synchronization (PTS), in which a team of cooperative agents obtains time periods of full communication mixed in with periods of limited communication. Also focusing on role assignment in the RoboCup, Gerkey and Mataric [26] detailed a role allocation procedure that instead casts the team problem as a series of iterated Optimal Assignment Problems (OAPs). Overall, this domain continues to provide a stable basis for evaluating both the problems found in MAS and their potential solutions. While this work has influenced the design of Defender, many of these algorithms have assumptions with regard to the number of agents known, periodic planning and communication times, etc. As such, algorithms designed for RoboCup will not be directly applicable to Defender. The work presented here removes a lot of these heavy assumptions, and thus it may be applied to the RoboCup.

There is another facet of abstract multi-agent models that intersects the RoboCup and predator-prey: target assignment problems. Arslan et al. [4] use a variant of regret matching and spatial adaptive play (SAP) in computing their pairings. Their work shows that these distributed algorithms tend to approximate the global optimum very closely. The research that followed adds to the strength of their argument that game theory may be applied to optimally determine pairings [44, 43, 54, 3]. While this work demonstrates the power of game theory in distributed domains, it differs from the work presented here in a number of ways. Primarily, the global objective function in Defender is constantly changing with respect to time. Additionally there are many more restrictions on the agent's field-of-view, movement, and observational accuracy. However, due to the popularity of regret matching in multi-agent systems as a whole, we have implemented it in Defender to gain another point of comparison. Gmytrasiewicz and Durfee [29, 28] created a problem similar to Defender entitled "air defense." It consisted of anti-air units trying to defend their territory. In their particular algorithm, they used the probability of successfully defending to determine each agent's action. Their problem was discretized, involved stationary agents, and their agents had reliable observations of the environment. While similar to Defender, these differences prevent their solution from being directly applied.

A recently formalized problem domain entitled *ah hoc teams* encapsulates the ideas surrounding heterogeneous agents that lack the knowledge of their teammate's decision-making processes. One of the original formal definitions was written by Stone et al. [82], in which they described a system that contained agents that could not directly communicate with one another but were able to adapt. Defender and our matrix game reconstruction algorithm fall within this new area of research. Specifically, we include evaluations on ad hoc teams in Defender to judge the adaptability of fictitious play and rational learning. The proceeding work by Stone et al. [81] builds on the formal ideas of ad hoc teams into the domains of the multi-armed bandit, as well as robotic soccer. Wu et al. [97] also experiment with ad hoc teams in Decentralized Partially-Observable Markov Decision Processes (Dec-POMDPs) that have limited communication. They found that even in their limited communication environment, a set of decent solutions may still be found.

In an effort to place a structure around the often complex situations found in MAS, *roles* and *tasks* are commonly defined by agent designers to consolidate and encourage the desired coordinated behavior. One of the earliest approaches to task allocation was by Smith [77], and his work on the Contract Net Protocol. In this distributed algorithm, contracts are formed and maintained by agents to complete tasks required by the community of agents. Expanded versions include ALLIANCE by Parker [65] and MURDOCH by Gerkey and Mataric [24]. These tend to be relatively straight forward auctions that are incredibly powerful and robust, so much so that they are among the most widely used algorithms in practice. A general framework called STEAM (Shell for TEAMwork) created by Tambe [84] uses this idea in a more general model and addresses the issues surrounding the robust mechanisms required for teamwork and cooperation. It bases its ideas on joint intention theory, in which agents commit to actions over their respective communication networks. We first looked towards these contract-based algorithms for our problem but quickly found that it required too many messages to be effective.

Gerkey and Mataric [25] provide a taxonomy for task allocation. They break down Multi-Robot Task Allocation (MRTA) into six categories, each defined by a tuple of three elements: single-task robots vs. multi-task robots, single-robot tasks vs. multi-robot tasks, and instantaneous assignment versus time-extended assignment. Stone and Veloso [79] decomposed their tasks into various layers, in particular they use formations to describe the structure of roles for agents, and subsequently their tasks at any given time. Their approach has proven to be one of the strongest in the RoboCup. While this type of decomposition through layers can help construct a robust network of roles, it also runs into issues when roles must be reassigned. Nair and Marsella [57] address this point as well as provide a mathematical model for role assignment in their Role-based Multiagent Team Decision Problem (RMTDP) framework. RMTDPs use Beliefs Desires and Intentions (BDI) along with POMDPs in their underlying structure to describe how multi-agent designers may determine the optimal selection of roles and role assignment algorithms for their particular problem. They tested their results in a number of problem domains, including RoboRescue, and even compared it with STEAM, with their framework successfully optimizing the allocation (and reallocation) of roles. Lastly, Campbell and Wu [14] provide a valuable analysis of the common types of role assignment, such as if the roles are predefined or dynamically created. In addition, their work spans a wide breadth of the role assignment literature, implying its importance in the future of MAS.

One common thread throughout all of multi-agent systems is in the design of cooperation and coordination. Jennings [40] postulated that MAS with both commitments and conventions can lead to emergent cooperation among the community of agents. Commitments are analogous to contracts between agents, and conventions are ways agents maintain these contracts in an ever-changing environment. Our work aligns itself with this hypothesis, claiming that in our selected problem domain, our method emerges with these commitments and conventions through fictitious play and rational learning in the dynamically changing game.

2.2.2 Predator-Prey and Pursuit-Evasion Problems

The original predator-prey problem was introduced by Benda et al. [7]. It consisted of four predators attempting to coordinate their movements to capture one prey moving randomly around a discrete toroidal grid-world. Once the predator agents have surrounded the prey on all four sides, the game ends. Figure 2.1 provides a visual example of the scenario. This simple agent cooperation problem has now become the standard for developing multi-agent algorithms. Predator-prey is encapsulated in the category of pursuit-evasion problems. The modern form of this broad area was developed by Parsons [67]. Pursuit-evasion problems entail pursuer agents seek to find and often capture evading agents. While it was originally designed with agents residing on a graph, it has since expanded to continuous domains. Our work on Defender expands on this original idea, bringing it into a continuous world with multiple prey and more restricted information for predators. We also include a haven [71] for the prey on the opposite side of the predators. This addition allows for new kinds of emergent behavior usually found in sports and other competitive environments.

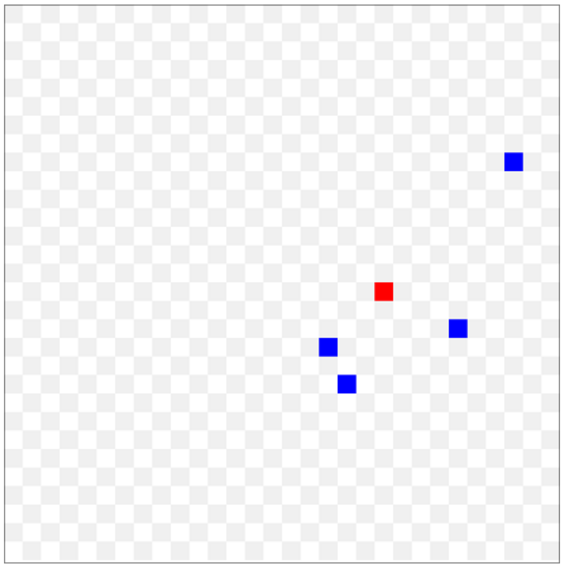


Figure 2.1: An example of the traditional predator-prey problem, as described by Benda et al. [7]. The blue squares represent the predators and the red square represents the prey.

Korf [48] applied a simple rule to predator-prey to construct emergent cooperation among the predator agents. The predators' objective was to decrease the distance between them and the prey while also increasing the distance between them and other predators. This configuration yields the desired "cross-shaped" formation structure. More recent work on predator-prey also has shown that teammate-aware agents obtain better results, as one might intuit [5]. Their work also focused on removing assumptions from the model, specifically the assumption in which predators know the behavior of the other predators on the field. They evaluated a variety of ad hoc algorithms that perform online adaptation and demonstrated team-aware agents are much more robust to the system dynamics. Denzinger and Fuchs [15] classified a subset of variants for the pursuit game and used genetic algorithms for the agent adaptation method. Undeger and Polat [85] investigated the effects of maze-like worlds on the predator-prey problem. They also included multiple prey in their simulations, demonstrating that their method for blocking escape directions worked quite well at catching the prey. The MICE platform has produced some interesting work on predator-prey [29, 28, 16, 55]. In particular, Durfee and Montgomery [16, 55] ran experiments covering multiple predators and multiple prey in their traditional environment. One key difference that they experimented with was situations in which the prey can reach a haven on the opposite side behind the predators. This obviously aligns itself with Defender, albeit their scenario's agents are supplied with much more information in comparison. Levy and Rosenschein [51] constructed a coalitional game for the predators to determine the utility for moving towards the prey versus blocking one of the escape directions. Their approach heavily used inter-agent communications among other assumptions, removing it as potential solution to our problem. These kinds of traditional predator-prey algorithms are very robust and produce excellent results in simulations, but our focus is on the effects of information-sparsity in multi-predator multi-prey games. Most of these algorithms require a great deal of unavailable knowledge to even be applied to our problem.

Predator-prey is a subproblem of the more general pursuit-evasion problem category. This focuses on a number of pursuers seeking to capture evaders within an environment. Originally, the problem was studied with one pursuer and one evader, and most often it was played on a graph (matrix or grid included) [18, 67]. There are however continuous time versions of the problem, such as the Homicidal Chauffeur Problem [18] that contains a faster, less agile pursuer and an agile, slower evader. Utgoff and Kashyap [86] applied fictitious play to the graph version and laboriously demonstrated some optimal pursuer behaviors that were viable in this situation. Their work also examined uncertainty to some extent, but differs from our work greatly in both problem statement and solution. Recently, Vidal et al. [90] created physical incarnations of their agents in an ad hoc team pursuit-evasion setting. Their pursuers sought to maximize a local or global objective function based on probabilities. Their real-world results showed that a globally-maximizing team of pursuers are able to find and capture the prey more efficiently in spite of the naive assumption that it is moving randomly. Gerkey et al. [27] create a ϕ-searcher algorithm that takes into account limited fields of view, formally evaluating the ϕ-searcher and the pursuit-evasion problem itself. Lastly, the introduction of a mediator agent was proposed by Keshmiri and Payandeh [46]. This agent serves as a point of communication among agents and a source of environmental knowledge. The mediator seeks to remedy the issues surrounding uncertainty by effectively assigning a strict role to the agent(s). The pursuit-evasion literature captures a wider range of solutions to combat uncertainty for our problem. The majority of the work is only partially applicable to Defender, but serves as another strong foundation to build upon.

Multi-agent research into pursuit-evasion and predator-prey lends itself nicely to robotic experimentation, e.g. in work done by Vidal et al. [90], especially those methods which use game theory. Skrzypczyk [76] examine the problem of target-based following in robots. They use single-stage matrix games, constructed individually by each agent, and solve for the Nash equilibrium and *min-max* solution to provide effective following behavior. Guibas et al. [31] worked towards a robotic simulation of visibility restrictions in the pursuit-evasion problem. While they only focus on a single pursuer, they do provide computational bounds on the model, in addition to simulations that reinforce their claims. Interestingly, Harmati and Skrzypczyk [33] use a semi-cooperative Stackleberg equilibrium in an attempt to partially obviate the issue of agents unknowingly moving out of globally optimal locations. Their work provided a robotic team simulation consisting of three robots. Our work focuses on the dynamics found in real world cooperative scenarios, for which robotic pursuit-evasion experimentation serves as point of comparison.

2.2.3 Multi-Agent Learning

At the intersection of game theory and multi-agent systems, one finds the work done on multi-agent learning. Shoham et al. [74] and Shoham and Keyton-Brown [73] compared various kinds of multi-agent learning algorithms founded in game theory, of which four have been selected for the work presented in this thesis. As previously described, fictitious play, rational learning, regret matching, and a number of others have all become prominent in the community. Some have also been custom-tailored for multi-agent systems, while others have been completely redeveloped. The paragraphs that follow detail some of the MAS-specific learning algorithms.

Rational learning [42] has been employed in stochastic domains in work done by Boutilier [10]. He showed that applying conventions in stochastic kinds of games allows for agents to properly converge even though they only receive information about the outcome of actions, not the action taken itself. While rational learning is popular due to Bayesian inference's probabilistic strength, regret-based learning [34] tends to be the most prevalent in the field. The primary reason for this is that in most perfect information games it converges to a strategy profile in which agents are always obtaining their best payoffs, i.e. a correlated equilibrium. That said, Jafari et al. [38] compared regret matching's performance over a collection of well-known games and showed that it failed to properly converge in even simple 2 player games.

Gradient ascent for iterated games (IGA) by Singh et al. [75] uses hill-climbing to iteratively solve a game in a similar way to that of reinforcement learning. Expanding on this, WoLF-IGA (Win or Learn Fast IGA) follows the iterated gradient ascent approach but changes the step size based on the performance of the current strategy. Bowling [11] used these ideas to create GIGA-WoLF, an algorithm that directly applied Generalized IGA to WoLF and proved that universal consistency (i.e. no regret) holds given all agents follow GIGA-WoLF.

One main area that this thesis explores is learning with time-varying utilities. Panait and Luke [64] describe it as "moving the goalposts," and claim it will be a vital part of future multi-agent learning algorithms. Early attempts to describe this dynamism used Temporal Difference

(TD), consisting of both a stochastic gradient ascent part and a parameter estimation part [6]. Brenner and Nebel [12] handle this varying payoff problem by breaking down agent decisions into planning and acting, allowing for algorithms to quickly interleave the two in various ways. Vidal and Durfee [89] tried to model and predict MAS by examining the error probabilities over time. Weinberg and Rosenschein [94] describes an algorithm for stochastic games that does not necessary converge to a Nash equilibrium. They instead focus on convergence to a best-response policy, so as to optimally behave in their dynamic games. Partially-Observable Stochastic games may potentially model time-varying utilities; however, most algorithms leverage the power of reinforcement learning methods, such as Q-Learning. This unfortunately is not mappable to our problem since any game states would usually not be repeated. Furthermore, there are particular realistic robot-like constraints, such as sensor noise and field-of-view, that are difficult to model.

Original models for multi-agent learning often focused on perfect game information [53, 93]. The more recent algorithms now focus on multi-agent learning in stochastic domains, the majority of which rely on some form of Markov Decision Processes (MDP). For example, some of the most important work in this area focuses on Decentralized Partially-Observable Markov Decision Processes (Dec-POMDPs) [8]. Bernstein et al. define it as a MDP with a set of observations provided to each agent at each stage and a method to distribute the system's received reward to each agent. Goldman and Zilberstein [30] further developed Dec-POMDPs into a set of classes based on types of observations and communication among agents. Nair and Tambe [58] proposed a hybrid BDI/POMDP framework called a Role-based Markov Team Decision Problem (RMTDP) in an effort to better capture learning in MAS. All of these approaches are viable means to describe the behavior in Defender except for the time-varying utilities in a non-discrete space and its particular uncertainties. We developed our role distribution games in order to capture these elements in a mathematical model.

2.2.4 Teammate Modeling

Multi-agent systems commonly run into the problem of modeling other teammates to predict their actions and adapt to them. To some degree, most multi-agent learning algorithms do this implicitly through their various belief updating routines. This subsection will focus more on the explicit modeling of other agents.

A well-known teammate modeling algorithm known as Recursive Modeling Method (RMM) was created by Vidal and Durfee [88], and was later expanded upon by Gmytrasiewicz and Durfee [29, 28]. The premise of their algorithm is to construct nested, often recursively structured models of other agents and prune away the implausible ones. This provides a collection of potential models so that when a decision must be made, the agent may do so quickly and effectively by incorporating only the most "important" models of others. Their approach was developed and tested in the air defense and predator-prey domains, demonstrating that it can indeed improve performance and maintain models of other agents.

Algorithms that create recursive structures of other agents are found throughout the field.

For example, Suryadi and Gmytrasiewicz [83] build decision models using inference diagrams and simulated their algorithm on the air defense domain, obtaining good results overall. Hu and Wellman [36] examined the depth to which agent's recursive trees were built, and demonstrate one level of recursive depth provides adequate results, with additional levels providing diminishing returns. Zettlemoyer et al. [100] use nesting beliefs in combination with Dec-POMDPs to filter incorrect or unimportant beliefs and to realize a better future reward in the partially observable environment. Latek et al. [49] also construct a recursive teammate modeling mechanism and run their algorithms on the Generalized Shapley Game. Bringing together microeconomic theory and multi-agent systems, they show that bounded rationality in this recursive modeling approach is directly related to limits on computation.

2.3 Communication Architectures

Communication methods are found throughout a wide range of fields. In this section we describe some of the previous research in the most relevant fields. The major focus of this section relates to our distributed communication architecture algorithm.

2.3.1 Computer Networking and Sensor Networks

The ALOHA packet system is one of the foundations of the networking community [69]. Created by Roberts, it works by simply transmitting a message on a shared channel, with collisions handled by waiting a random amount of time before retrying. Both slotted and non-slotted versions of the algorithm were proposed. This has been used in everything from cell phone network's common control channel in GSM to satellite-based internet protocols [37]. Our distributed communication architecture (DCA) will be an improvement over this approach. It uses a similar randomization method but differs in that it is attempting to build a distributed, adaptive communication architecture among agents. Specifically, the *slot selection state* in our algorithm is the most similar component, with the *learning state* and *settled state* operating differently. The ALOHA packet system has been extended over the years [1] to improve performance and include new features; however, the core of the algorithm has remained constant.

Fair queuing is another well-known approach to scheduling [56, 87]. The idea is to provide a weighted queuing system based on how many packets or messages a particular node is sending. Therefore, nodes send their messages in such a way that the queues fill in an evenly distributed way. This kind of behavior is often found in round-robin scheduling methods, too. *Round-robin scheduling* has also been a popular topic in the networking community [41, 92, 32, 61, 39, 9]. In particular, Hahne [32] created a straightforward round-robin scheduling algorithm and proved the worst-case bounds on it. Mainly, the research clearly showed that round-robin techniques approximate max-min fair rates, especially as the data's window size is increased. Our communication algorithm centers around the distributed construction of a round-robin pattern, but to our knowledge the work in the networking community always leaned towards heavily centralized systems.

Sensor networking is wide-ranging topic that overlaps with multi-agent systems and computer networking. For the sake of brevity, we turn the reader to a strong survey paper on the topic by Akyildiz et al. [2]. While sensor networks are distributed like MAS, their development strives to create true networks. As a result, broadcast networks are not often investigated. We would however like to mention two recent papers that represent some of the time-slot-based research. Rhee et al. [68] created an algorithm entitled DRAND which focuses on allocating time slots based on the location of the sensors in relation to their neighbors. The messages are either accepted or rejected at nodes to hopefully signal the proper path for the sending node to transmit on. Lin et al. [52] introduced GLASS, an extension of DRAND that vastly improves performance. Still, their slot allocations are based on neighboring sensor nodes, whereas we consider a full group of agents broadcasting to one another in a round-robin manner.

2.3.2 Communications in Multi-Agent Systems

Distributed artificial intelligence and multi-agent systems constantly deal with communication in the operation of their agents. Smith [77] created the Contract Net Protocol which effectively distributes contracts among agents in the system. The benefit to distributed contract sharing over a communication medium is that there is no centralized control, i.e. a single point of failure. The communication in the Contract Net Protocol was done via direct agent-to-agent messaging. The actual framework used the standard low-level routing tables, a network packet structure, etc. to perform the message passing. Expanding on this idea, Parker [65] created an algorithm called ALLIANCE. ALLIANCE included contract degradation that handled failed attempts by agents to meet the contract specifications by freeing the contract for another agent to perform it. It also focused on task impatience and acquiescence scores to encourage contract-taking and communication. Perhaps the most recent expansion was developed by Gerkey and Mataric [24, 23]. They focused their communication architecture on a publish/subscribe model. Their idea was to send messages addressed by content rather than addressing messages to agents. This subject-based addressing uses message broadcasts on a network; agents who are listening for a particular message type may either listen or ignore any messages received. They argue this kind of content-centered broadcasting approach mitigates the issues in unicast messaging since it is much more inefficient for multi-agent communication. The work presented in this thesis uses a kind of broadcasting system and could be thought of as a content-based messaging approach.

The RoboCup [47, 62] often produces very interesting challenges in the realm of communication methods. The competition itself relegates all robots on the field to share a single communication channel. It is for this reason that we decided to focus on single channel low bandwidth communication for our approach as well. Stone and Veloso [79] examine this PTS domain and create a communication paradigm that incorporates: identifying teammate messages versus opponent messages, adapting to interference created by hostile agents, preventing agents from “talking all at once,” handling lost messages, and determining how to maximize shared information. Their approach is not feasible for our problem because of the differences in problem requirements, e.g. Defender has unknown team sizes *a priori* and is not a Periodic Time Synchronization (PTS) scenario. There are of course many other examples of robotic control communication systems that address these common issues [101, 99].

A large subset of communication in MAS is centered around agent communication languages and speech-act theory. Speech acts are tools for cooperative systems that formally describe the ways in which agents are able to convey information to one another [73]. Agent communication languages are the practical implementation of speech act theory. One of the most popular agent languages is called Knowledge Query and Manipulation Language (KQML) Finin et al. [17]. It facilitates inter-agent knowledge sharing through their *performatives* that define the speech acts available to agents. The Multiagent Planning Language (MAPL) is another popular one that focuses instead on a language that allows agents to rapidly formulate plans [12]. Our communication algorithm tends to lie in the mid-level messaging issues; it is a mix of *what* may be sent and *how* to send it. Xuan et al. [98] also examine this middle ground. They argue that *when* to communicate is a decision that an agent must make, just as much as what to send. This stems from the idea that communications have a cost associated with them, and is supported by the literature as it favors a blend of game theory and stochastic processes to describe the systems.

2.4 Summary

In this chapter, we have provided a survey of the related literature in the fields of multi-agent systems, game theory, agent-centered learning, teammate modeling, and communication algorithms. Each of these areas and the papers referenced within relate to the main ideas covered in this thesis. From this survey of the state of the art we have learned the trends in the field. Multi-agent learning has shifted from a traditional game theory domain to a more uncertain knowledge, imprecise observation, and ad hoc team domain. Predator-prey and pursuit-evasion problems have also tended towards more complicated scenarios that require cooperation under very limited information. Communication architectures now commonly must handle optimal message passing and simultaneously allow for arbitrary collections of agents to effectively convey ideas. The culmination of these trends in the literature has led us to pursue Defender and our distributed communication architecture algorithm. In the chapters that follow, we detail our solutions to these problems currently faced in the state of the art.

Chapter 3

Preliminary Material

This chapter covers the essential foundation for the material discussed in the later chapters. We begin by defining multi-agent systems and the traditional predator-prey problem. While we primarily describe these for completeness, the concepts will be discussed again in chapter 4. Next we discuss the basics of game theory, namely: normal form games, Nash equilibrium, learning algorithms, and the concept of partial observability. These main ideas will be used throughout chapter 4. The section that follows switches to a brief introduction of slot delimited communication, which forms the core of our communication algorithm detailed in chapter 5.

3.1 Multi-Agent Systems

3.1.1 Towards a Formal Definition

While there have been many attempts to formalize multi-agent systems, all have struggled to fully encompass the breadth of the problem without remaining too nebulous. A similar issue arose when researchers sought a formal definition for an agent. What degree of autonomy is required for an entity to be called an agent [96]? Should there be limitations on communication capabilities since full and limitless communication might trivialize it to a single-agent system [80, 64]? For the purposes of this thesis, we refine our view to the following pair of definitions for agents and multi-agent systems.

Definition 1. Agents *are autonomous entities that react to and interact with the environment in which it resides. An agent is able to interact with other agents through either direct or indirect communication.*

Definition 2. Multi-Agent Systems *(MAS) describe a collection of autonomous agents that reside in an environment and seek to optimize some local and/or global objective. Each agent has a limited view of the environment, individual objectives which may or may not align with other agents, and/or limitations placed on communication with other agents.*

With these two definitions we are able to describe the problems and solutions presented in this thesis. The next subsection provides a formalized example of a multi-agent system: the predator-prey problem.

3.1.2 The Predator-Prey Problem

As described in chapter 2, the predator-prey problem was originally proposed by Benda et al. [7]. We provide it here for completeness and so that it might be more easily compared to Defender.

Definition 3. The Predator-Prey Problem *is a tuple* (N, M, E)*, such that:*

- E *is a toroidal grid-world environment of discrete size* s *by* s *and discrete time taken in stages.*
- N *is a set of* n *predator agents, traditionally* $n = 4$*, that are randomly placed in* E *at initialization. The movement action that is performed at each stage is taken from the set* $\{N, S, E, W, \emptyset\}$*.*
- M *is a single prey agent, randomly placed in* E *at initialization, that randomly takes movement actions at each stage taken from the set* $\{N, S, E, W, \emptyset\}$*.*
- *Agents in* N *and* M *are not able to move into an occupied cell.*
- *The agent update order is randomly determined at the start of each stage.*
- *The predators in* N *must coordinate to surround the prey for the game to end, i.e. force the prey's set of possible movement actions to be* $\{\emptyset\}$*.*

In spite of its simplicity, it is a powerful tool to test cooperative algorithms. The average time until the prey is captured, any emergent behavior that is observed, and the restrictions on the particular algorithm are often used to measure the overall effectiveness. We also use these measures in our experiments with Defender.

3.2 Game Theory

Games are a structure to describe the interactions among players or agents. We begin by describing the normal form (or strategic form) of a game then continue by describing various learning algorithms and how to define limited knowledge.

3.2.1 Normal Form and Solution Concepts

There are two commonly used forms to represent games: normal form and extensive form. While there are equivalences between the two, normal form tends to be used in the description of a single stage game, whereas extensive form is often used to describe multiple "steps" of play. Repeated games mostly leverage normal form to define players interacting in the same game multiple times, so we will narrow our view to normal form games exclusively. We refer the reader to the books by Shoham and Keyton-Brown [73] and Fudenberg and Tirole [22] for a more in-depth look on the subject. We would like to state that the notation and definitions we have used in this thesis, while universal, were largely influenced by these two books.

Definition 4. Normal form games *(also called strategic form games) are defined as a tuple* (N, A, U)*, such that:*

- N *is a set of* n *players (agents).*
- $A = A_1 \times \ldots \times A_n$ *is a class of action sets, in which* A_i *is the set of all actions available to player* $i \in N$*.*
- $U = (u_1, \ldots, u_n)$ *is the set of utility (payoff) functions, such that* $u_i : A \mapsto \mathbb{R}$ *for* $i \in N$*.*

Definition 5. *We call the set* $a = (a_1, \ldots, a_n)$*,* $a \in A$ *an* **action profile***. Frequently it will be denoted as* (a_i, a_{-i}) *for some player* $i \in N$ *and action* $a_i \in A_i$ *with all other players* $-i = N \setminus \{i\}$ *and their actions* $a_{-i} \in A_{-i} = A_1 \times A_{i-1} \times A_{i+1} \times A_n$*.*

Definition 6. *A* **mixed strategy** $s_i = (p_1, \ldots, p_k)$*,* $k = |A_i|$*, for player* $i \in N$ *is defined as the set of a discrete probability distribution (or equivalently non-negative weights) over the set of actions* $A_i = (a_1, \ldots, a_k)$ *with* $\sum_{i=1}^{k} p_i = 1$*.*

Definition 7. *A* **pure strategy** s_i *for player* $i \in N$ *is a special case of a mixed strategy for which all of the weight is placed on one action, i.e.* $s_i = (p_1, \ldots, p_j, \ldots p_n)$ *with* $p_j = 1$ *and* $p_k = 0$ *for all* $k \neq j$*. Note that playing a pure strategy is equivalent to playing an action.*

Definition 8. *A* **strategy profile** $S = (s_1, \ldots, s_n)$ *denotes a particular class of strategies taken from all players* $i \in N$*.*

The distinction between pure and mixed strategies is important. There are a few interpretations of mixed strategies despite having the same effect for a game. One interpretation is that the probabilities assigned to a player's actions in a mixed strategy determine the proportions (weights) that the player plays that action. For MAS this makes sense in contexts for which roles are equivalent to actions. For example, if the roles are forager and protector, it makes sense that one agent might act as a mix of the two in any situation. Another interpretation is that these describe a kind of distribution over the player's actions. It follows that anytime the game is played, the players will randomly select their action based on this distribution, and end up playing only one action fully at a particular stage. If the game were to be repeated over a large number of stages, then the distribution of actions performed by each player should intuitively converge to their mixed strategy. We will see in a later section how mixed strategies relate to learning in games.

Notation 1. *For a mixed strategy* s *for player* $i \in N$ *and action* $a \in A_i$ *we denote* $p(a)$ *as the probability of action* a *as given by* $p_j \in s_i$ *such that* j *corresponds to the action* a*.*

Notation 2. *For the purposes of this thesis, we will only consider action profiles instead of strategy profiles. Therefore, all strategy profiles $S = A_i$ for all $i \in N$. However, the definitions below will hold for any mixed strategy profiles as well. Mixed strategy profiles will only be used for describing probability distributions over actions.*

We now are able to define what are known as *solution concepts*, the most popular of which is Nash equilibrium. Solution concepts state a possible "solution" to a game with respect to the particular quantity the concept seeks to optimize. In other words, the various solution concepts that exist each attempt to describe a kind of optimal configuration of strategies. There is no solution concept that fully defines the notion of optimality for *every* situation, but Nash equilibrium has risen to be the standard point of comparison among them.

Definition 9. *Given a player $i \in N$ and a set of action profiles $a_{-i} \in A_{-i}$ for all other players, a* **best response** *is a action a_i such that $u_i(a_i, a_{-i}) \geq u_i(a'_i, a_{-i})$ for all other $a'_i \in A_i$, $a_i \neq a'_i$.*

Definition 10. *A* **Nash equilibrium** *is a action profile a such that for all players $i \in N$, a_i is a best response given a_{-i}.*

This powerful idea actually has a variety of elegant mathematical properties underlying its behavior. Since we have simplified the definition for action profiles alone, we have developed a customized algorithm to compute the Nash equilibrium for a specific kind of game. Algorithm 3.1 assumes that all players have the same number of actions. It uses an abstracted matrix game that has uniquely indexed rows that represent each cell of the normal form matrix game. The columns in this abstracted matrix represent a particular player. Thus a row and column pair corresponds to a payoff for a player in a specific cell.

One of the downsides with Nash equilibrium is that it does not necessarily model scenarios for which agents are not behaving rationally. With the goal of applying these solution concepts in games that are repeated over many stages, we will examine a concept known as *regret*. As described in chapter 2, *no regret* learning algorithms converge to a state of no regret. The idea is to simply subtract what the agent obtained at a stage by the utility that it could have obtained if it had played optimally. Note that this definition of regret is always non-negative. Taking this one step further, the agent is able to compute the action or strategy that minimizes its maximum regret possible. This method is called minimax regret. The formal definition of regret and minimax regret are given below.

Definition 11. **Regret** *for the action $a_i \in A_i$ of an agent $i \in N$, given all other agents are playing action profile $a_{-i} \in A_{-i}$, is defined as $\max_{a'_i \in A_i} u_i(a'_i, a_{-i}) - u_i(a_i, a_{-i})$.*

Definition 12. **Minimax regret** *for player $i \in N$ is defined as:*

$$\arg\min_{a_i \in A_i} \Big[\max_{a_{-i} \in A_{-i}} \Big((\max_{a'_i \in A_i} u_i(a'_i, a_{-i})) - u_i(a_i, a_{-i}) \Big) \Big].$$

Finally, we have supplied an example of the game constructed in our algorithm for Defender: the anti-coordination game. This kind of game appears in a multitude of problems. It describes

Algorithm 3.1 *computeNashEquilibrium*: Computes the pure strategy Nash equilibria.

```
Require: $n$: The number of players.
Require: $m$: The number of actions for a player.
Require: $M \in \mathbb{R}^{m^n \times n}$: The abstract matrix game, in which a row corresponds to a cell, and a
    column corresponds to a player's payoff for that cell.
equilibria ← $\emptyset$
for $i \in \{0, \ldots, m^n - 1\}$ do
    $a \leftarrow \emptyset$
    for $j \in \{0, \ldots, n - 1\}$ do
        $a \leftarrow a \bigcup \{resolveAction(n, m, i, j)\}$
    end for
    $best \leftarrow$ true
    for $j \in \{0, \ldots, n - 1\}$ do
        $start \leftarrow \lfloor i - a_j * m^j \rfloor$
        $stop \leftarrow \lfloor start + m^{j+1} \rfloor$
        $step \leftarrow \lfloor m^j \rfloor$
        for $k \in \{start, start + step, start + 2 * step, \ldots, stop\}$ do
            if $M_{kj} > M_{ij}$ then
                $best \leftarrow$ false
            end if
        end for
    end for
    if $best =$ true then
        $equilibria \leftarrow equilibria \bigcup \{i\}$
    end if
end for
return $equilibria$
```

a situation in which agents only receive a payoff if they successfully select different actions from other agents.

Definition 13. *An* **anti-coordination game** *is an n-player game for which any players $i, j \in N$ for which the utility $u_i(a_i, a_{-i}) = u_j(a_j, a_{-j}) = 0$ for action profiles with $a_i = a_j$, and utilities greater than zero otherwise. For example, a 2-player anti-coordination game has the payoff matrix:*

$$\begin{array}{cc|c|c|}
 & & \multicolumn{2}{c}{Player2} \\
 & & A & B \\
\hline
Player1 & A & 0,0 & \alpha, \beta \\
\hline
 & B & \gamma, \delta & 0,0 \\
\hline
\end{array}$$

with player set $N = \{1, 2\}$, action sets $A_1 = A_2 = \{A, B\}$, and utility function returning 0 and $\alpha, \beta, \gamma, \delta > 0$ as described in the normal form above.

3.2.2 Learning Algorithms

In the previous section, we hinted at the idea of repeating a game over many stages. Below we provide the formal definition of a repeated game. These games allow us to finally begin modeling problems like Defender, whereby players must learn over time to adapt to the shifting strategies of others.

Definition 14. *A* **repeated game** *is a tuple* (N, A, U, T) *such that* $G = (N, A, U)$ *is a normal form game, and* $T : (G, \Omega) \mapsto (G, \Omega')$ *is a stage transition function that maps a game to itself with updated observations* $\Omega \subset \Omega'$ *from previously played games. We call each repeated play of the game a* stage.

Notation 3. *We denote* Ω *to be the list of observations,* Ω_i *to be the list of observations made by player* $i \in N$*, and* Ω_{ij} *to be the list of observations made by player* $i \in N$ *of player* $j \in N$*.*

Repeated games often are examined with an average reward, possibly discounted over many stages. We will not examine this aspect in detail because our application will be in a dynamic MAS environment. With the concept of a repeated game, learning algorithms may be formally defined over the list of observations. In other words, with this stationary game the players continue to play one another and build their observations over time.

Also, note that our learning algorithms are described in their *action-based* versions, i.e. each performs the argmax operation to determine the action. The alternative would be to construct a probability distribution over each action and randomly select an action over that distribution. We opted against this for a number of computational and practical reasons. Most importantly, we observed the probability-based versions did not produce emergent commitment to actions, and had much worse results. This decision will be discussed again in chapter 4.

Fictitious play assumes that all other players are playing a stationary mixed strategy. Therefore, the observed distribution of their actions taken over many stages should approximate their true distribution. While knowingly naive, this assumption enables a simple representation for selecting the action at a stage by maximizing the expected utility. Equations 3.1 and 3.2 describe this approach.

Definition 15. Fictitious play *players at stage* t *select their action* a^* *following:*

$$a^* = \underset{a_i \in A_i}{\arg\max} \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \prod_{a_j \in a_{-i}} p_i(a_j) \tag{3.1}$$

with:

$$p_i(a_j) = \frac{1}{|\Omega_{ij}|} \sum_{\omega \in \Omega_{ij}} \mathcal{I}\{\omega = a_j\} \tag{3.2}$$

Rational learning is a similar method to fictitious play that selects an action through utilizing the maximum expected utility. This approach instead models the distribution of other player's actions (i.e. their mixed strategy) through a Bayesian updating mechanism performed at each stage. We will assume *Dirichlet priors* throughout this thesis, but any prior distribution is technically viable. The reason we selected the Dirichlet distribution is because it is the conjugate prior for the categorical distribution. Therefore, a Dirichlet prior for Bayes' rule in combination with a categorical distribution likelihood yields a Dirichlet distributed posterior. The categorical distribution is, in effect, the probability distributions over the observations of other players. So the difference from fictitious play lies in its update mechanism [10]. Furthermore, the Dirichlet distribution with $\vec{\alpha}$ set to one yields the discrete uniform distribution, i.e. the same probability for each category. This is our initial prior. For more information on these distributions, please see Appendix A. The probability distribution update step follows Bayesian inference, as shown in equations 3.3 and 3.4.

Definition 16. Rational learning *players at stage t select their action a^* following:*

$$a^* = \arg\max_{a_i \in A_i} \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \prod_{a_j \in a_{-i}} p_i(a_j) \tag{3.3}$$

with:

$$p_i(a_j) = \frac{p(\Omega_{ij}|a_j)p(a_j)}{\sum_{a_k \in a_{-i}} p(\Omega_{ik}|a_k)p(a_k)} \tag{3.4}$$

The last of the learning algorithms we examine extends upon idea of regret. Regret matching selects the action that minimizes the regret formed by the payoff obtained at this stage t and the payoff the player could have obtained at t if he had played optimally at each stage $1, \ldots, t$. This is then averaged by the regret over all actions to obtain the proportions. This proportion for each action is the probability that the action is performed at stage t. Note that unlike the previous definition of regret, since we are dealing with averages we take the maximum of the regret value and zero to ensure regret is always non-negative.

Definition 17. Regret matching *players at stage t select their action a^* probabilistically following:*

$$p(a^*) = \frac{\max\{r_i(a_i) - r_i, 0\}}{\sum_{a' \in A_i} \max\{r_i(a') - r_i, 0\}} \quad \forall a_i \in A_i \tag{3.5}$$

with:

$$r_i(a') = \frac{1}{|\Omega_{ij}|} \sum_{t=1}^{|\Omega_{ij}|} u_i^{(t)}(a') \qquad \textit{and} \qquad r_i = \frac{1}{|\Omega_{ij}|} \sum_{t=1}^{|\Omega_{ij}|} u_i^{(t)}(a_i^{*(t)}) \tag{3.6}$$

These three learning algorithms are not the only kind, as evidenced by those described in chapter 2. Our rationale for selecting these are two fold. First, these are among the most popular learning algorithms from the game theory and multi-agent learning communities. Second, the algorithms are not necessarily reliant on a high number of stages or a large quantity of information to approximate adequate solutions; they are simple, reactive, responsive, and highly adaptable. In the proceeding chapters, we will examine the effects of time-varying payoffs on these algorithms and their unique application to predator-prey problems such as Defender.

3.2.3 Partial Observability and Uncertainty

With the goal of applying these algorithms in highly dynamic environments, we consider the concepts of partial observability and uncertainty. In game theory, there are generally three ways to describe "haziness" in a game: incomplete information, imperfect information, and imperfect recall. They may be synthesized to describe some idea of partial observability and uncertainty for our problem.

Definition 18. *A game which has* **incomplete information** *means that players in the game are missing information about the game structure and/or the payoff functions of other players.*

Definition 19. *A game which has* **imperfect information** *means that players in the game lack knowledge of the actions other players could or will take.*

Definition 20. *A game which has* **imperfect recall** *is a game in which players do not always know how they got to a particular state (or stage). This is often described in the context of extensive form games.*

Let us use Defender as an example to better illustrate these concepts. Defender is an incomplete information game, since the intersection of two predator agent's field-of-view may contain some shared prey that are observed. The non-intersecting portions may contain prey unobservable by both. Hence, the game for both may lack information found in the *true* "global game." Defender is also an imperfect information game, since each predator that observes another predator does not necessarily know the history of actions that form the observed predator's decision-making process at that stage. Finally, Defender is an imperfect recall game, since so little is known about the true global game at any stage, and no agent can discern *precisely* how the system itself arrived at that stage. These concepts will be more clear once the formal definition of Defender is stated in chapter 4.

3.3 Tracking

Most applications of multi-agent systems require some form of tracking algorithm, whether it be for external objects, such as a person, or internal components, such as an accelerometer's output. Specifically, robotic forms of predator-prey and pursuit-evasion problems require tracking of the various agents within the environment. Tracking methods are also found in the RoboCup and RoboRescue domains for obvious reasons. This section briefly touches on the basics of tracking: the Extended Kalman Filter (EKF) and gating mechanisms.

3.3.1 Extended Kalman Filter

The Extended Kalman Filter (EKF) is a widely used approach in the tracking literature. It is based on the Kalman Filter [45], which iteratively takes imprecise measurements of the state and statistically filters the noise leaving a prediction of the true state of the system. The original Kalman Filter is only optimal for linear state transitions, so the EKF was designed to remedy this by approximating the non-linearity at each stage through a locally linearized version of state transition. For this subsection, we followed the equations described by Welch and Bishop [95].

First we will define each variable for use in the equations. The current true state at time $t > 0$ is denoted by $x_t \in \mathbb{R}^n$, with the *a priori* state estimate denoted by $\hat{x}_t^-$ and the *a posteriori* state estimate denoted by $\hat{x}_t$. The function f describes the transformation from state $t-1$ to state t, often highly non-linear in real-world applications. Similarly, we let $z_t \in \mathbb{R}^n$ be the true measurement taken at time t, $\hat{z}_t^-$ denotes the *a priori* measurement estimate, and $\hat{z}_t$ denote the *a posteriori* measurement estimate. The function h then describes the measurement taken from a state with noise included. We will let $\vec{w}$ be the noise found in the state and $\vec{v}$ be the noise found in the measurement. Lastly, the matrices A, W, H, and V are the Jacobians of $\frac{\partial f}{\partial \vec{x}}$, $\frac{\partial f}{\partial \vec{w}}$, $\frac{\partial h}{\partial \vec{x}}$, and $\frac{\partial h}{\partial \vec{v}}$, respectively.

The EKF may be broken into two steps. The first may be thought of as a *prediction step* for which the *a priori* state is estimated as well as its error from noise. The second is considered an *update step* which first computes the *Kalman gain* K_t, then updates the *a posteriori* state and noise based on the measurements taken. Equations 3.7 and 3.8 define the prediction step, and equations 3.9, 3.10, and 3.11 define the update step.

$$\hat{x}_t^- = f(\hat{x}_{t-1}, u_{t-1}, 0) \tag{3.7}$$

$$P_t^- = A_t P_{t-1} A_t^T + W_t Q_{t-1} W_t^T \tag{3.8}$$

$$K_t = P_t^- H_t^T (H_t P_t^- H_t^T + V_t R_t V_t^T)^{-1} \tag{3.9}$$

$$\hat{x}_t = \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-, 0)) \tag{3.10}$$

$$P_t = (I - K_t H_t) P_t^- \tag{3.11}$$

We use the EKF described here to reduce the tracking noise for the observed state of other agents. This state will include location, orientation, and speed. We will not attempt to cover any of the extraneous material on the subject, as this approach is ubiquitous throughout the literature for reducing sensor noise in tracking situations.

3.3.2 Score and Threshold Gating Mechanisms

In algorithms that track multiple agents, ideally a single track is assigned to each agent. This track represents the history of observations for that particular agent's state, i.e. location, orientation, and speed. An issue arises in cases for which tracks begin to overlap. In this scenario, some mechanism must exist to pair new observations with their corresponding tracks. Figure 3.1 provides a visual example of this problem.

In the example figure, the new observation must be either paired with track a or track b, or it may require a new track to be created. To handle this problem, we use *score and threshold based gating*. Threshold gating simply uses some distance metric to pair observations with tracks. Formally, for a list of new observations Ω_{new} we may apply some distance metric such as the Euclidean norm for each track's recorded history of observations. Let there be k tracks, and denote this history as $\Omega_1^{(t-1)}, \ldots, \Omega_k^{(t-1)}$. Thus, for all $i \in \{1, \ldots, k\}$, a prediction for the track's next state ω_i^*, and a threshold value T:

$$\Omega_i^{(t)} = \Omega_i^{(t-1)} \bigcup_{\omega \in \Omega_{new}} \begin{cases} \omega & \text{if } \|\omega_i^* - \omega\|_2 < T \\ \emptyset & \text{otherwise} \end{cases} \tag{3.12}$$

Obviously, this alone will not suffice because it might include more than one track observed at each time step, and observations may be shared among tracks in close proximity to one another. Score-based gating solves this issue by assigning a score to each track and pairing new observations to current tracks either by greedily taking the top score for each one or using the Hungarian Method. Tracks that do not receive a new observation may be removed and additional new observations without a track may prompt the creation of a new track. Formally, with k tracks, the history for track $i \in \{1, \ldots, k\}$ up until time $t-1$ given by $\Omega_i^{(t-1)}$, the predicted new observation's state ω_i^*, and the score function $s : \omega \mapsto \mathbb{R}^n$ for $\omega \in \Omega_{new}$:

$$\Omega_i^{(t)} = \Omega_i^{(t-1)} \bigcup \left[\arg\max_{\omega \in \Omega_{new}} s(\omega) \right] \tag{3.13}$$

Score-based gating assigns new observations to tracks based on this score function $s(\omega)$. We combine threshold-based gating and score-based gating to first prune the set of new observations that are unlikely candidate observations, then assign the optimal configuration of tracks based on their scores. The tracker in Defender actually produces some of the most interesting behavior for the agent's learning algorithms, since tracks are removed and added *on-the-fly*. This causes the game size and observation history to change dramatically. We will see the effects of the tracker in chapter 4.

– Include if you want to force it to be fixed to the paragraph.

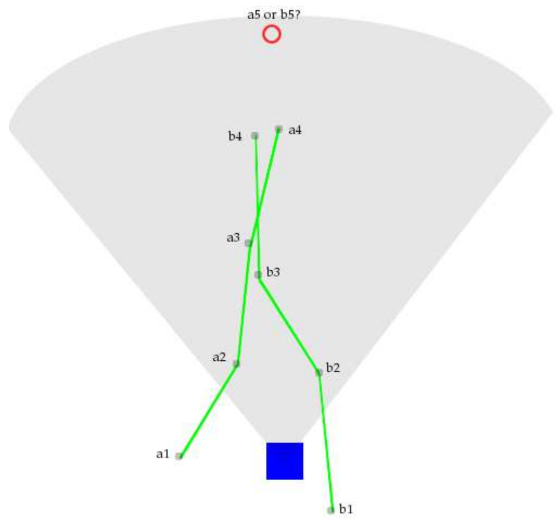


Figure 3.1: An example of the gating problem encountered in tracking multiple agents. The blue square and grey cone represent an agent with a limited field-of-view. The green lines represent the two tracks' state observation histories. The red circle represents a new observation.

3.4 Communication

Multi-agent communication is a tool used by agents to share local information with other agents in the system. As mentioned in chapter 2, one commonly used approach of handling multiple individuals in a network is to use slot-based architectures. In this section, we provide an explanation of these systems and their application to broadcast networks.

3.4.1 Channels and Broadcasting

For completeness, we first provide a high level overview of broadcasting and the idea of channels. To begin, a *broadcast system* describes an architecture in which agents are able to send messages to a subset of the other agents in the system simultaneously. This is in contrast with the other extreme: a *unicast system*, in which messages are only sent from one agent to one other agent. Broadcast systems are frequently seen in multi-agent systems, for example the RoboCup limits communication to broadcasts on a single channel. For the purposes of this thesis, a channel will be synonymous with a frequency. This is not always the case because often both time and frequency are used in conjunction to define a channel. Therefore, a single channel system defined this way implies that messages must be sent in a structured manner over time. The communication algorithm described in chapter 5 is designed for single channel broadcast systems. It creates this structured pattern of message sending over time, allowing for agents to broadcast information within the team of predators.

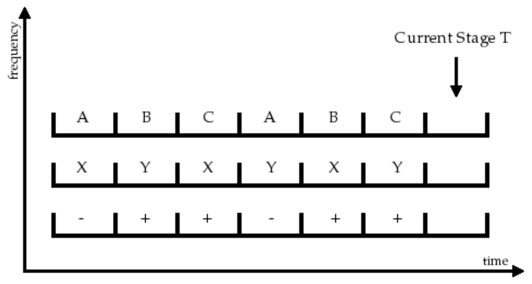


Figure 3.2: An example of slot-based communication in both time (x-axis) and frequency (y-axis) domains for a single system.

3.4.2 Slot-Based Communication

Slot-based algorithms focus on discretizing the continuum of frequency and time into what are called *slots*. Slots describe a window in which one particular agent may send a message. Figure 3.2 demonstrates how slot based communication may be used in both the time and frequency domains. As the figure suggests, a slot may be described as a pair of integers denoting the frequency and time. The top two channels (on the frequency axis) have agents $\{A, B, C\}$ and $\{X, Y\}$ communicating, respectively. The lowest channel has two agents $\{+, -\}$ that are communicating such that agent $+$ sends two messages for every one sent by agent $-$. Our distributed communication architecture strives to create a pattern akin to the topmost channel in figure 3.2.

3.5 Summary

This chapter has covered the relevant background material for the topics of this thesis. It has also set the notation for the chapters that follow. We began this chapter by introducing the ideas and challenges of multi-agent systems, which encapsulates the work presented in this thesis. Next, the framework of game theory and learning in games was introduced. The application of these concepts in highly dynamic multi-agent systems is the primary focus of our investigation. We then outlined the basics of broadcast and slot-based communications since our distributed communication algorithm lies within that area. The chapters that follow will now use this material in the explanation of our methods.

Chapter 4

The Defender Problem

In this chapter, we present Defender, a multi-agent problem that introduces uncertainty and requires rapid cooperation among agents. We propose solutions founded in game theoretic learning, e.g. fictitious play, rational learning, etc., with the additional payoff function and tracking components. Each potential method will be thoroughly tested in a series of simulations that vary the number of agents and environmental parameters. We also examine the behavior of each algorithm in ad hoc teams and improve the computational time of the top performing algorithms. Finally, we conclude the chapter with a discussion of our findings and how it relates to the field as a whole.

4.1 Description

Uncertainty will play a key role in the development of new multi-agent systems, as is evident from the current state of the art discussed in chapter 2. Most of the current algorithms manage this kind of uncertainty through contract management [77, 65, 24], or attempt to mitigate its effects over long periods of iteration, as reinforcement learning methods do in Dec-POMDPs [8, 30, 58]. These approaches are not readily applicable to Defender due to their time or communication requirements.

The multi-predator multi-prey environment depicted in Defender was constructed to describe scenarios found throughout MAS applications. Defender itself models a situation similar to a defensive line in football, soccer, or rugby. These kinds of cooperative role assignment scenarios are therefore found in the RoboCup [47] soccer competitions. Our goal is to build algorithms leveraging game theory that create rapid commitments among agents and learn to adapt amidst very unreliable information. Predator-prey is one of the standard testing platforms for multi-agent algorithms. For this reason, we designed Defender to be a predator-prey problem. We took ideas of a haven for prey from both sports as well as preliminary work on predator-prey in graphs [67, 71]. Variants akin to Defender also include Cooperative Multi-Target Observation of

Multiple Moving Targets (CMOMMT) [66] and experiments done on the MICE platform [16, 55], among many others.

4.2 Formal Model

Here we define Defender and standardize the notation used in later sections.

Definition 21. The Defender Problem *consists of parameters which describe the particular scenario and environment. They may be written as the tuple (N, M, E, T).*

- *N is a set of n predator agents, detailed in section 4.2.1.*
- *M is a set of m prey agents, detailed in section 4.2.1.*
- *E is a set of parameters that describes the environment, detailed in section 4.2.2.*
- *T is the transition method that defines how agents update at each iteration, mapping $T : \{N, M\} \mapsto \{N, M\}$, detailed in section 4.2.3.*

4.2.1 Agents

A *state* of Defender consists of a particular instance of agents $N \bigcup M$. Each agent i at any stage $t > 0$ is defined by the parameters: $\{\vec{p}_i, \vec{v}_i, \vec{a}_i, \theta_i, \Omega_i, L_i\}$. The three vectors $\vec{p}_i$, $\vec{v}_i$, and $\vec{a}_i$ denote the agent's position, velocity, and acceleration, respectively. The scalar variable $\theta_i \in [0, 2\pi)$ describes the orientation of the agent in radians. The set Ω_i is a list of observations made, for which Ω_{ij} denotes the list of observations made by agent i of agent j, if any. The tracker determines how new observations are added to Ω_i and only contains location, velocity, and orientation. Formally, for an observation $\omega \in \Omega_i$, denote the observation's location, velocity, and orientation as $\vec{p}_\omega$, $\vec{v}_\omega$, and θ_ω. Lastly, L_i denotes the decision-making algorithm used by the agent. For predators, this may be: fictitious play, rational learning, regret matching, minimax regret, or a greedy strategy. For prey, the possibilities are: simple or maneuvering. For an agent i, we will denote the desired velocity to be $\vec{v}_{i,desired}$ and the desired orientation to be $\theta_{i,desired}$ for use in the decision-making algorithm.

4.2.2 Environmental Parameters

We define a set of parameters E to place limitations on the environment and agents: $\{s^*, c^*, t^*, \phi^*, (\delta_{loc}, \delta_{vel}, \delta_\phi), (p_{miss}, p_{err}), (a_{vmax}, a_{tmax}, a_{update})\}$. Following the literature on predator-prey problems, the area is a two-dimensional, continuous, fixed world of size s^* by s^*. Agents begin on their corresponding "goal" sides opposite one another with each goal being $\frac{1}{15}s^*$ in height. Initially, they are randomly assigned a location in their goal region $\vec{p}_i =< \mathcal{U}(0, s^*), \mathcal{U}(0, \frac{1}{15}s^*) >$ for $i \in N$ and $\vec{p}_j =< \mathcal{U}(0, s^*), \mathcal{U}(\frac{14}{15}s^*, s^*) >$ for $j \in M$. Their initial direction follows $\mathcal{U}(-\delta_\phi, \delta_\phi)$, and begin play after $\mathcal{U}(0, t^*)$ seconds.

Predators $i \in N$ may capture a prey $j \in M$ once $\|\vec{p}_i - \vec{p}_j\|_2 < c^*$. Their goal is to capture as many prey as possible before the prey reach the opposite side, at which point the game ends; this emulates the mechanics found in sports. Agents are restricted to a limited field-of-view given by ϕ^* such that for agent i, observations are made only in the visual window on $(\theta_i - \frac{\phi^*}{2}, \theta_i + \frac{\phi^*}{2})$. Furthermore, an agent's observations are subject to sensor noise as found in robotic applications. Specifically, for an observation ω and the true values for the observation ψ, $\vec{p}_\omega = \vec{p}_\psi + < \mathcal{N}(0, \delta_{loc}), \mathcal{N}(0, \delta_{loc}) >$, $\vec{v}_\omega = \vec{v}_\psi + < \mathcal{N}(0, \delta_{vel}), \mathcal{N}(0, \delta_{vel}) >$, and $\theta_\omega = \theta_\psi + \mathcal{N}(0, \delta_\phi)$. Agents also have a predefined maximum speed a_{vmax}, turn speed a_{tmax}, and update interval a_{update}; this simulates the computational burdens often faced in robotic multi-agent systems.

Agents do not carry any knowledge with them from game to game. Furthermore, agents must *infer* the actions of other agents; they may not be simply "known" once the agent is observed. This constraint again models the very common multi-agent problems. While we do not use communication methods in this section, we will discuss the application of inter-agent message passing in chapter 5.

4.2.3 The State Transition

At each stage, agents move in the environment following their velocity and acceleration. Agents i adjust their $\vec{v}_{i,desired}$ and $\theta_{i,desired}$ only during their update times, not continuously. A friction constant $\mathcal{F}$ and an acceleration rate $\mathcal{R}$ have also been introduced to the model, since this behavior is found in both robotic MAS and the RoboCup. For our simulations, we let $\mathcal{F} = 0.96$ and $\mathcal{R} = 0.75$. The following set of ordered equations defines the state transition made continuously in Defender.

$$
\begin{aligned}
&\vec{v}_i \leftarrow \vec{v}_i + \vec{a}_i \\
&\vec{p}_i \leftarrow \vec{p}_i + \vec{v}_i + \frac{1}{2}\vec{a}_i^2 \\
&\vec{v}_i \leftarrow \mathcal{F} \times \vec{v}_i \\
&\vec{a}_i \leftarrow \vec{a}_i + sign(\|\vec{v}_{i,desired}\| - \|\vec{v}_i\|) \times \mathcal{R} \times \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}
\end{aligned}
$$

The state transition also includes the comparison of predator-prey pairs to check for capture, as described in section 4.2.2. Finally, it also performs the check to see if a prey $j \in M$ has reached the blue side, also described in the previous section. Conversely, if all prey are successfully captured by the predators, then the game ends.

4.3 The Game Structure

In any optimization problem, the objective function often plays the biggest role in performance. This is compounded in distributed systems since the system operates on a function that is potentially missing information. The payoff function in game theory also has this same effect. For Defender, we focus on an *anti-coordination game* as defined in chapter 3. The game structure rewards agents only for strategy profiles in which they are the sole agent performing a particular action. Contrary to its name, this game allows us to build an emergent coordination and cooperation behavior.

4.3.1 Formal Statement

We define the normal form of predators at any given stage to follow definition 22. Diverging from traditional game theory, we allow the payoff functions to vary over time. We will describe the result of this decision in section 4.8.

Definition 22. Defender's normal form *at stage t is a game consisting of a tuple (N, A, U) where:*

- *N is the set of n predators.*
- *$A = (A_1, \ldots, A_n)$ with each A_i being a set of prey (actions) available to predator i, not necessarily all observable by predator i.*
- *$U = (u_1, \ldots, u_n)$ is a set of functions, with each u_i representing the payoff (or utility) an agent receives for a action profile a.*

The payoff function is given by equation 4.1. It follows almost directly from definition 13 for an anti-coordination game. Note that the function $u_i(a_i)$ in the equation below returns the payoff only with respect to action a_i. This will be useful in the optimizations found in section 4.6.

$$u_i(a) = \begin{cases} u_i(a_i), & \text{if } \forall 1 \leq j \leq n,\, j \neq i,\, a_i \neq a_j \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

The function $u_i(a_i)$ will often provide larger payoffs for the prey $a_i \in A_i$ that are both reachable and closer to agent i. Initially, we considered many potential functions such as the probability of reaching the prey [4], the probability of blocking escape routes [85], and even simple Euclidean distance [43]. In our initial assessment of each approach, we determined that the *estimated distance* provided the most accurate measure of an agent's utility. This kind of payoff function has been both popular and successful in other multi-agent systems [48, 43, 5, 90]. Furthermore, probabilistic methods often use an estimated distance in their computation anyway [4, 90]. Equation 4.2 states our utility formula and algorithm 4.1 describes the exact computation of this distance.

$$u_i(a_i) = clamp(p_{max} - d * clamp(0.85 + 0.05 * (b - a), 0.7, 1.0), 0, p_{max}) \tag{4.2}$$

The variables used in equation 4.2 are detailed below.

- p_{max} is a constant for the maximum payoff possible from the definition of Defender; for example, in robotic incarnations it might be proportional to the maximum sensor range.
- d is the distance estimate of the desired path to travel, computed in algorithm 4.1.
- b is the number of other observed predators who have shorter distance estimates, performing algorithm 4.1 centered at these predators.
- a is the number of additional potential prey that might exist if a miss occurs, that again uses algorithm 4.1 to count the number of visible prey (projected locations using their track information) after reaching the estimated path intersection point.

For the *estimatedPath* algorithm, we must briefly define two small helper functions. The first function, $enforceTurnSpeed(\theta)$, takes the difference between the θ desired and the current simulated theta and ensures the θ value does not change by more than A_{tmax}. The second function, $enforce(\theta, \vec{p}_{prey}, \theta_{prey})$ enforces the constraint on θ that it does not turn the current simulated predator beyond its field-of-view as given by ϕ^*. This is crucial so that an agent does not lose line-of-sight and subsequently drop the prey from its internal game. Algorithm 4.1 describes the procedure in detail, and figure 4.1 provides a visual example.

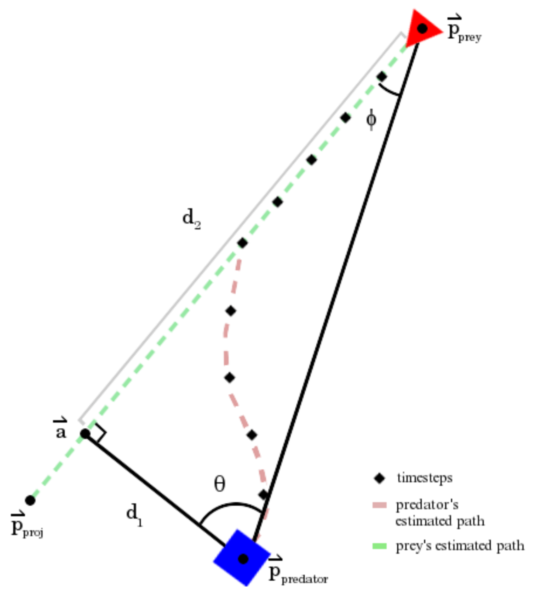


Figure 4.1: A visual example of the estimated path algorithm. The variables contained are found in algorithm 4.1.

4.3.2 An Example

To get a sense of the run-time operational structure, we provide a simplified example. The main point of this is to draw attention to the final game structure, which emerges as a strange kind of

Algorithm 4.1 *estimatedPath*: Estimated ideal path algorithm which is used in both the payoff and tracker functions.

```
Require: $\omega_{predator}$: The observation of a predator with components: $\vec{p}_{predator}$, $\vec{v}_{predator}$, and $\theta_{predator}$.
Require: $\omega_{prey}$: The observation of a prey with components: $\vec{p}_{prey}$, $\vec{v}_{prey}$, and $\theta_{prey}$.
Require: $step$: The step size; a smaller number increases prediction granularity.
 1: $\vec{r} \leftarrow \emptyset$
 2: $\vec{r}_0 \leftarrow \emptyset$
 3: $d_{total} \leftarrow 0$
 4: while $\|\vec{p}_{predator} - \vec{p}_{prey}\|_2 > capture$ do
 5:     $d_1 \leftarrow \frac{|(\vec{p}_{proj} - \vec{p}_{prey}) \times (\vec{p}_{prey} - \vec{p}_{predator})|}{|\vec{p}_{proj} - \vec{p}_{prey}|}$
 6:     $\psi \leftarrow \arcsin\left(\frac{d_1}{\|\vec{p}_{prey} - \vec{p}_{predator}\|}\right)$
 7:     $d_2 \leftarrow \frac{d_1}{\tan(\psi)}$
 8:     $\vec{a} \leftarrow \vec{p}_{prey} + d_2 \begin{bmatrix} \cos\theta_{prey} \\ \sin\theta_{prey} \end{bmatrix}$
 9:     $\theta \leftarrow \arctan\left(\frac{\vec{a}.y - \vec{p}_{predator}.y}{\vec{a}.x - \vec{p}_{predator}.x}\right)$
10:     $\theta \leftarrow enforceTurnSpeed(\theta)$
11:     $\theta_{predator} \leftarrow enforceFacePrey(\theta, \vec{p}_{prey}, \theta_{prey})$
12:     $\vec{p}_{predator} \leftarrow \vec{p}_{predator} + step \times \|\vec{v}_{predator}\| \times \begin{bmatrix} \cos(\theta_{predator}) \\ \sin(\theta_{predator}) \end{bmatrix}$
13:     $\vec{p}_{prey} \leftarrow \vec{p}_{prey} + step \times \|\vec{v}_{prey}\| \times \begin{bmatrix} \cos(\theta_{prey}) \\ \sin(\theta_{prey}) \end{bmatrix}$
14:     $\vec{p}_{proj} \leftarrow \vec{p}_{prey} + \sqrt{2}s^* \times \begin{bmatrix} \cos(\theta_{prey}) \\ \sin(\theta_{prey}) \end{bmatrix}$
15:     $\vec{r} \leftarrow \vec{p}_{predator}$
16:     if $\vec{r}_0 = \emptyset$ then
17:         $\vec{r}_0 \leftarrow \vec{p}_{predator}$
18:     end if
19:     $d_{total} \leftarrow d_{total} + \|\vec{v}_{predator}\|$
20: end while
21: return $\vec{r}, \vec{r}_0, d_{total}$
```

partially observable anti-coordination game. This n player anti-coordination game is ordered by how agents observe one another. It is a result of having $u_i(a_i, a_{-i}) \geq 0$ and $u_j(a_j, a_{-j}) = 0$ for two predators i and j, $i \neq j$, in which $a_i = a_j$ but agent i does not observe predator j. To better illustrate this, figure 4.2 presents a scenario which generates the normal form shown in table 4.1.

Each cell (an action profile) receives its payoffs according to equations 4.1 and 4.2. For example, agent 2's payoff for action profile $a^* = (A, B, C)$ is $u_2(a^*_2, a^*_{-2}) = u_2(B, \{A, C\}) = clamp(p_{max} - d * clamp(0.85 + 0.05 * (b - a), 0.7, 1.0), 0, p_{max})$ with $b = 1$, and $a = 0$. So, $u_2(B, A, C) = clamp(10 - 4 * clamp(0.85 + 0.05 - 0, 0.7, 1.0), 0, 10) = clamp(10 - 4 * 0.9, 0, 10) = 6.4$.

Upon further examination of this matrix game, it is clear that it does exhibit an anti-coordination game in the construction of one agent's payoffs. While the lack of full information does not necessarily allow for true anti-coordination, consider the case where all agents had full

$2 : a_2^*$ \ $3 : a_3^*$	A	B	C
A	8.3, 0, 0	8.3, 0, 4.3	8.3, 0, 2.4
B	8.3, 6.4, 0	8.3, 6.4, 0	8.3, 6.4, 2.4

$1 : a_1^* = A$

$2 : a_2^*$ \ $3 : a_3^*$	A	B	C
A	7.45, 7.3, 0	7.45, 7.3, 0	7.45, 7.3, 2.4
B	7.45, 0, 5.25	7.45, 0, 0	7.45, 0, 2.4

$1 : a_1^* = B$

$2 : a_2^*$ \ $3 : a_3^*$	A	B	C
A	6.6, 7.3, 0	6.6, 7.3, 4.3	6.6, 7.3, 0
B	6.6, 6.4, 5.25	6.6, 6.4, 0	6.6, 6.4, 0

$1 : a_1^* = C$

Table 4.1: The normal form of the game depicted in figure 4.2.

knowledge of the true game. For an agent, it would consist of payoffs proportional to the distance metric for action profiles lacking other agents selecting the same opponent, and a zero payoff in the cases that they did. This is definitely a kind of anti-coordination game; however, in practice the limited field-of-view prevents its realization.

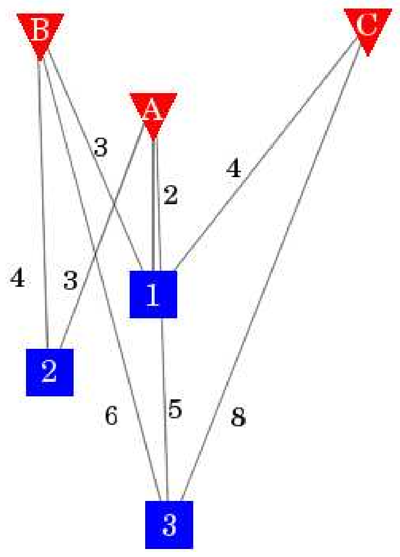


Figure 4.2: A simplified example of three predators and three prey to explain Defender's payoffs using its normal form. Agent 1 observes no defenders and opponents $\{A, B, C\}$, agent 2 observes defender $\{1\}$ and opponents $\{A, B\}$, and agent 3 observes defenders $\{1, 2\}$ and opponents $\{A, B, C\}$.

4.4 The Tracker

As previously stated, realistic applications quite often rely heavily on a tracker. The tracker might employ vision-based tracking routines to track moving objects or may be entirely driven by sensor information. In Defender's construction we assumed a robotic application akin to the RoboCup. Thus, the tracking errors closely mimic those found in vision-based tracking, e.g. random loss of track and miss classification. This section describes the tracker and touches on the effects that this dynamic tracker has on the internal game of each predator.

4.4.1 Extended Kalman Filter and Gating

As in every real-world system, during the update procedure the agents receive new information about other agents in the environment. These observations are subject to noise, modeled as the Normal distribution detailed in definition 21. The Extended Kalman Filter (EKF) is used to approximate the true observation information, following equations 3.7-3.11.

The tracker also uses threshold and score-based gating to pair tracks and new observations together. We apply the thresholds $T_{\vec{p}}$ and T_θ for position and direction, respectively. Algorithm 4.2 describes this in detail.

Algorithm 4.2 *tracker*: The tracker function that maintains the tracks and new observations.

```
Require: $\Omega_i$: The current list of observations for predator $i \in N$. Each $j \in \Omega_i$ has the components: $\vec{p}_j$, $\vec{v}_j$, and $\theta_j$ denoting the result of the EKF over the observations. Additionally, it includes the variable $\alpha_j$ denoting the age, i.e. the number of stages since the track last received a new observation.
Require: $\Omega_{new}$: The new observations, each $\omega \in \Omega_{new}$ has the components: $\vec{p}_\omega$, $\vec{v}_\omega$, and $\theta_\omega$.
1: for $j \in \Omega_i$ do
2:   if $\alpha_j > 1$ then
3:     // The following statement removes old tracks.
4:     $\Omega_i \leftarrow \Omega_i \setminus \{j\}$
5:   end if
6:   $best \leftarrow 1$
7:   $\omega_{best} \leftarrow \emptyset$
8:   for $\omega \in \Omega_{new}$ do
9:     $score \leftarrow \frac{1}{2}\left(\frac{\|\vec{p}_j - \vec{p}_\omega\|_2}{T_{\vec{p}}} + \frac{\|\theta_j - \theta_\omega\|_2}{T_\theta}\right)$
10:    if $\|\vec{p}_j - \vec{p}_\omega\|_2 \leq T_{\vec{p}}$ and $\|\theta_j - \theta_\omega\|_2 \leq T_\theta$ and $score < best$ then
11:      $best \leftarrow score$
12:      $\omega_{best} \leftarrow \omega$
13:    end if
14:  end for
15:  $\Omega_{new} \leftarrow \Omega_{new} \setminus \{\omega\}$
16:  $\Omega_{ij} \leftarrow \Omega_{ij} \bigcup \{\omega\}$
17: end for
18: // The following adds new tracks, given $\Omega_{new}$ contains the unassigned observations.
19: $\Omega_i \leftarrow \Omega_i \bigcup \Omega_{new}$
```

The algorithm uses fairly common mechanisms for assigning observations to tracks. The more interesting outcomes are a result of the addition and subtraction of tracks. In a later section, we cover the effects of the tracker as it pertains to the game structure.

4.4.2 Action Estimation for Predators

The tracker also must perform the estimations of which action observed predators selected. This estimator function simply selects the action that maximizes the observed predators' utilities given the observed prey. Equation 4.3 below describes this operation, leveraging equations 4.1 and 4.2. In the equation, we let $i, j \in N$, predator i is estimating which prey predator j is currently chasing, using all prey $k \in M$ observed by i denoted as Ω_{ik}. Therefore, the action a_j represents predator j's action at the current stage.

$$a_j = \arg\max_{\omega \in \Omega_{ik}} u_j(\omega) \tag{4.3}$$

The reason that this is a reasonable estimate relies on the assumption that the observed location, orientation, and speed of the predator implies the action it has selected. The construction of our action selection and the time interval between updates actually provide time for the predator to orient and move toward its selected prey. This may then be observed by other predators at the next update stage. Obviously, there is the potential for errors in estimation from either a lack of knowledge or observation noise. In chapter 5 we will apply communications to this predator-prey model in an attempt to mitigate errors resulting from false-positive estimations.

4.4.3 System Behavior

As previously stated, the tracker's removal of old tracks and addition of new tracks cause drastic changes to the game structure during play. Furthermore, this action estimator introduces new noise into the system. Overall, there are four main effects: a jagged observation history, a potential for resetting the history, the spontaneous addition and removal of both agents and actions, and erroneous action estimation for predators.

A "jagged" observation history means that the length of the set of observations differs for each observed track. During the computation in our learning algorithms, this often will cause inconsistencies among the probability distributions or regret over actions. For example, consider a predator that observes two other predators: A and B, and two prey: 1 and 2. Observed predator A may have only two observations recorded, whereas observed predator B might have ten observations recorded. Therefore, the prey selections include a more complete picture of one observed predator versus the other.

The observation history may also be reset as a result of agents moving out of view then back into view. Once an agent leaves the field-of-view, the corresponding track is removed. If this same agent reenters the field-of-view, it is treated as a completely new agent. On one hand, the loss of the true history is beneficial since the scenario might have drastically changed in the intervening time. Thus, any legacy observations could actually hinder judgment. On the other hand, the loss of this history could have been crucial to cement the probability distribution over the observed agent's actions. In either case, it does provide a multitude of interesting results that will be discussed again in section 4.8.

The tracker's constant pruning of tracks causes very interesting behavior in the matrix game representation. The matrix actually changes in dimension; new prey add new rows for each predator, and new predators add a new dimension based on the number of prey. Our approach ignores these changes and relies on the steady growth of observation histories to ease the abrupt changes. We will continue our discussion of this behavior in the next section.

The erroneous action estimation is the last primary effect that the tracker causes on our game. In the tracker's estimation of a predator's selected prey, we may falsely estimate the incorrect prey. This is usually the case when the predator may not be able to observe the true prey being chased. This causes false observations which are used in the learning algorithms. This uncertainty is handled by relying on the dynamic nature of the game. Our approach assumes either the prey will become visible or the predator will chase the prey outside the field-of-view, causing the predator to be removed from the internal game.

Tracking other agents in the system causes these four kinds of behavior. These issues might have been managed to some extent with more complicated game theoretic models, such as Bayesian games or partially-observable stochastic games. Instead, we will show that our approach actually results in the desired emergent agent behavior. The proceeding sections will now investigate the selected learning algorithms, optimizations, and experimentation.

4.5 Learning Algorithms

We focus our approach's decision-making component on the well-known game theoretic algorithms defined in chapter 3. Fictitious play and rational learning are related in that they attempt to model a predator's mixed strategy distributions; however, they each update their beliefs differently. Regret matching and minimax regret both use the concept of regret, but regret matching includes observations over multiple stages, whereas minimax regret examines only a single stage. Lastly, we use a simple greedy strategy as a baseline comparison.

For completeness, we provide the framework used by all predators to select their action during the stage in algorithm 4.3. One important aspect of this algorithm lies in its desired velocity formula, the final line in the algorithm. It reduces the predator's speed slightly as a function of the number of visible predators. We have found that this heuristic slows the changes between each internal game and provides the much needed time for the predator to assess the actions of others. The more predators that are visible, the more information must be processed to select a reasonable prey. We denote a_i^* to be the result from one of the learning algorithms we selected. Let $a_{i,current}$ be the current prey the predator (i) is chasing. Finally, let the $chase(a_{i,current})$ function prompt the predator to chase the prey given by $a_{i,current}$ until the next update stage.

Algorithm B.1 describes our action-based fictitious play implementation. It follows equa-

Algorithm 4.3 *update*: The update step for a predator to decide which prey to chase.

```
Require: $\Omega_{new}$: The new observations, each $\omega \in \Omega_{new}$ has the components: $\vec{p}_\omega$, $\vec{v}_\omega$, and $\theta_\omega$.
Require: $k$: The number of visible predators.
1: $tracker(\Omega_i, \Omega_{new})$
2: $a_{i,current} \leftarrow \arg\max_{a_i \in A_i} L_i(a_i)$
3: $chase(a_{i,current})$
4: // Slow down slightly as a function of the number of visible predators.
5: $\vec{v}_{i,desired} \leftarrow \frac{\vec{v}_{i,desired}}{\|\vec{v}_{i,desired}\|} A_{vmax}(1 - \min(k * 0.1, 0.5))$
```

tion 4.4. Closely related to fictitious play, our rational learning predator logic is given in algorithm B.2. It implements equation 4.5. Both of these predators have the same general algorithmic structure stated in algorithm 4.4, which is optimized in section 4.6.

$$L_i(a_i) = \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \prod_{a_j \in a_{-i}} \frac{1}{|\Omega_{ij}|} \sum_{\omega \in \Omega_{ij}} \mathcal{I}\{\omega = a_j\} \tag{4.4}$$

$$L_i(a_i) = \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \prod_{a_j \in a_{-i}} \frac{p(\Omega_{ij}|a_j)p(a_j)}{\sum_{a_k \in a_{-i}} p(\Omega_{ik}|a_k)p(a_k)} \tag{4.5}$$

Our regret-based predators are more distinct than the previous two predators described above. Regret matching predators follow algorithm B.3 created from equation 4.6, and minimax regret predators implement algorithm B.4 which is summarized by equation 4.7.

$$L_i(a_i) = \frac{\max\{r_i(a_i) - r_i, 0\}}{\sum_{a' \in A_i} \max\{r_i(a') - r_i, 0\}} \tag{4.6}$$

$$r_i(a') = \frac{1}{|\Omega_{ij}|} \sum_{t=1}^{|\Omega_{ij}|} u_i^{(t)}(a') \qquad \text{and} \qquad r_i = \frac{1}{|\Omega_{ij}|} \sum_{t=1}^{|\Omega_{ij}|} u_i^{(t)}(a_i^{*(t)})$$

$$L_i(a_i) = -\Big[\max_{a_{-i} \in A_i} \Big((\max_{a'_i \in A_i} u_i(a'_i, a_{-i})) - u_i(a_i, a_{-i}) \Big) \Big] \tag{4.7}$$

The last implementation is the baseline greedy predator given by algorithm B.5. This implementation resolves which prey to chase by equation 4.8.

$$L_i(a_i) = u_i(a_i) \tag{4.8}$$

When we initially built the algorithms, we tried using mixed strategies (i.e. probabilistic action selection). Understandably, we observed very poor performance for all of them. The reason comes from the pure randomness and time spent between updates; there was no true commitment on behalf of a predator to chase a prey. So we decided to implement action-based learning algorithms instead. We immediately observed vast improvements over all learning algorithms when we made the change. This result will be understood more easily after section 4.7

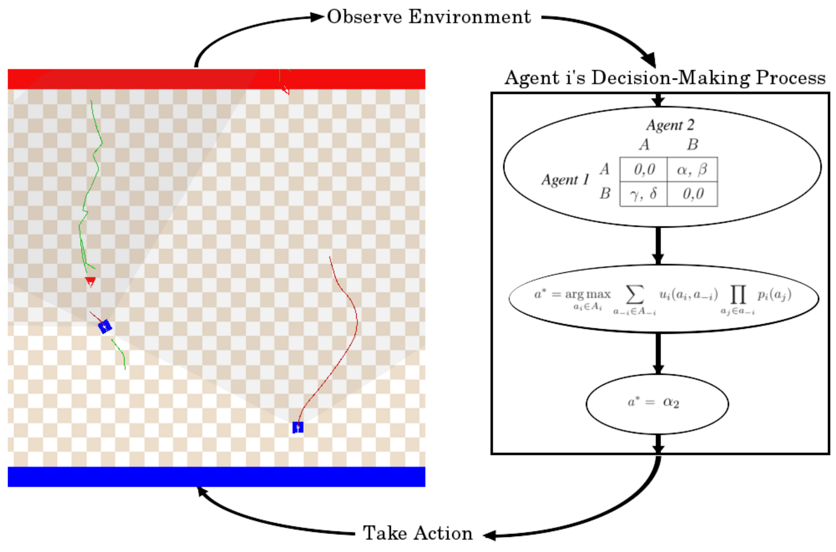

Observe Environment
Agent i's Decision-Making Process
Agent 2
A B
Agent 1
A 0,0 α, β
B γ, δ 0,0
$a^* = \arg\max_{a_i \in A_i} \sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \prod_{a_j \in a_{-i}} p_i(a_j)$
$a^* = \alpha_2$
Take Action

Figure 4.3: A visual control flow diagram for Defender's learning algorithms.

and discussed in detail as part of section 4.8.

For this to emulate a traditional application of game theory, our main objective was to approximate a stable game. Since our utilities and observations change constantly, we sought to perform updates fast enough so that from the learning algorithm's perspective the game does not change much between stages. This control flow for our learning algorithms is visually explained in figure 4.3. Essentially, at any given update step (i.e. stage) each agent receives observations from the environment, updates their internal anti-coordination game, updates their probability distributions and other pertinent variables, runs their learning algorithm, and chases the resultant prey.

4.6 Computational Complexity and Optimizations

During the implementation of the algorithms described in the previous section, we encountered some scaling issues. This is a common problem in multi-agent systems and is a direction for future research [64]. To improve the scalability of our overall approach, we leverage the particular game structure of anti-coordination games to reduce the computational complexity of our agent's decision algorithms. The addition of fictitious play and rational learning's mathematical formulation enable a specific algorithmic to exploit the game's quantity of zeros and improve run-time operation.

This section begins with a basic complexity analysis of each algorithm. Then, we delve into the properties of fictitious play and rational learning by breaking the equation into two segments. Using this knowledge, we continue by solving for the closed-form equation of the number of zeros multiplied during the computation. Next, we write a general optimized algorithm and state its new computation time. We close the section with an experiment for validation and a summary.

4.6.1 Complexity Analysis

Fictitious play and rational learning's complexity may easily be found from their respective equations in chapter 3. Let us denote n to be the number of observed predators, and m to be the number of observed prey. Computing the *argmax* requires a total of m steps; the summation requires a number of operations equation to the total quantity of action permutations for observed predators which is m^n. The inner product finally requires n steps to complete. Therefore, fictitious play and rational learning follow a worst-case complexity of $O(nm^{(n+1)})$.

Note that computing the probability for each action may instead be computed *in situ.*

$$p_j^{t+1}(a) \leftarrow (|\Omega_{ij}^t| p_j^t(a) + \omega_{ij}^{t+1}) / |\Omega_{ij}^{t+1}|$$

Regret matching is much simpler, since the computational burden is instead placed on memory. It requires a constant regret computation, effectively scaling computation by two. The *argmax* (in the action-based version) requires m steps and the denominator only needs to be computed once at another m steps. Thus the worst-case complexity of regret matching is just $O(m^2)$. Unfortunately, minimax regret suffers from a similar complexity problem as fictitious play and rational learning because it must also iterate over all permutations of action profiles. It requires m steps for the *argmin* component, m^n for the outer *min* component, and m steps for the innermost *max* component. Thus, the total worst-case complexity for minimax regret is $O(m^{(n+2)})$.

In summary, regret matching is much more tractable to perform during run-time. Minimax regret suffers greatly from the addition of more agents, as does fictitious play and rational learning. Of course, the simple greedy approach only takes $O(n)$, so it is technically the most computationally efficient in the worst-case. The sections that follow seek to optimize fictitious play and rational learning to achieve vastly improved performance and subsequently improved scalability.

4.6.2 Properties of Fictitious Play and Rational Learning

As suggested by the introduction, fictitious play and rational learning will be our main focus. They will be shown to produce good results in section 4.7. We will begin by analyzing the properties of the learning algorithms to find an equation that can be optimized in its corresponding algorithmic form.

The equation for maximizing expected utility has been used in the definition of both action-based fictitious play and rational learning. The action selected a^* at each stage is given in equation 4.9.

$$a_i^* = \arg\max_{a_i \in A_i} \Big[\sum_{a_{-i} \in A_{-i}} u_i(a_i, a_{-i}) \prod_{a' \in a_{-i}} p_i(a') \Big] \tag{4.9}$$

We many rewrite this equation by separating the payoff function into its two parts.

$$a_i^* = \arg\max_{a_i \in A_i} \Big[\sum_{\substack{a_{-i} \in A_{-i} \\ a_i \notin a_{-i}}} u_i(a_i) \prod_{a' \in a_{-i}} p_i(a') + \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \in a_{-i}}} 0 \prod_{a' \in a_{-i}} p_i(a') \Big]$$

Since $u_i(a_i)$ no longer depends on a_{-i}, we bring it outside the summation.

$$a_i^* = \arg\max_{a_i \in A_i} \Big[u_i(a_i) \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \notin a_{-i}}} \prod_{a' \in a_{-i}} p_i(a') + 0 \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \in a_{-i}}} \prod_{a' \in a_{-i}} p_i(a') \Big]$$

For notational clarity, let us denote $p_i^*(a_i)$ as follows.

$$p_i^*(a_i) = \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \notin a_{-i}}} \prod_{a' \in a_{-i}} p_i(a')$$

Now we have the final solution as defined in equation 4.10 below.

$$a_i^* = \arg\max_{a_i \in A_i} \Big[u_i(a_i) p_i^*(a_i) \Big] \tag{4.10}$$

To better understand this equation, let's turn to the two main parts: $u_i(a_i)$ and $p_i^*(a_i)$. The first function is easy to understand; $u_i(a_i)$ is proportional to the expected path distance from this predator (i) to the observed prey a_i. The value is slightly modified based on how many other agents might reach the prey faster and how many prey would be available if the predator fails to capture the prey after an attempt. The second part of equation 4.10 is most easily understood by breaking down $p_i^*(a_i)$.

$$\begin{aligned} p_i^*(a_i) &= \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \notin a_{-i}}} \prod_{a' \in a_{-i}} p(a') \\ &= \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \notin a_{-i}}} p(\text{the event that } a_{-i} \text{ occurs}) \\ &= p(\text{the event that other predators are} \\ &\qquad \text{not selecting the prey } a_i) \end{aligned}$$

So fictitious play and rational learning, in their optimized expected utility form (equation 4.10), decide their prey based on *maximizing the inverse distance to travel times the probability no other observed predator is also chasing that prey.*

4.6.3 Quantity of Zeros

We would like to know the proportions of "$u_i(a_i)$" to "0" as part of the summation. In the next section, we will use this knowledge to build an optimized algorithm. We compute the

0 1 2	Zero?	0 1 2	Zero?	0 1 2	Zero?
A A A	Yes	B A A	Yes	C A A	Yes
A A B	Yes	B A B	Yes	C A B	Yes
A A C	Yes	B A C	Yes	C A C	Yes
A B A	Yes	B B A	Yes	C B A	Yes
A B B	Yes	B B B	No	C B B	No
A B C	Yes	B B C	No	C B C	No
A C A	Yes	B C A	Yes	C C A	Yes
A C B	Yes	B C B	No	C C B	No
A C C	Yes	B C C	No	C C C	No

Table 4.2: The tabular example for the computation in $p_i^*(a_i)$ with $n = m = 3$. Each cell corresponds to a permutation and each letter in that permutation corresponds to the selected prey A, B, or C of observed predators 0, 1, and 2. From equation 4.10, we examine the case in which the *argmax*'s parameter a_i is A, and denote a *Yes* if an observed predator in a permutation cell also selects A.

number of zeros overall by examining each individual calculation in $p^*(a_i)$. Table 4.2 provides a visual structure to aid in our explanation. As the table suggests, there are a lot of zeros in this computation (resulting from the payoff function). For each of these zeros, that particular element in the permutation's summation (equation 4.10) does not need to be computed. We take advantage of this by skipping the ones that would result in an addition by zero. This is the "$a_i \in a_{-i}$" part of the equation.

Our internal algorithm used in fictitious play and rational learning iterates over this entire table. The pseudocode for the general case is found in algorithm 4.4 below. Our list's order may be traced back to the arbitrary results from the tracker's observations made during each previous time step. The function $index(x)$, where $x \in X$ and X is a list, returns the index value of the element x in the list (zero-indexed).

It turns out that the number of zeros in the summation may be determined from this logic. There is always the same number of action "conflicts" regardless of the selection of a_i and a predictable pattern in the algorithm's construction. Algorithm 4.5 computes the number of zeros given n and m. Using table 4.2 as a reference, it begins by counting the number of cells that predator 0 selected A. The number of remaining "blocks" of the same size is $m-1$, and for each of those blocks the other predators are then counted in a similar manner. As a further level of verification, we ran the algorithm itself with a counter for the number of zeros encountered and found the numbers matched exactly.

We may expand algorithm 4.5 into its mathematical form, so that a closed-form solution can be found. These steps are provided below.

$$\begin{aligned} z &= zeroCount(n, m) \\ &= m^{n-1} + (m-1)\Big[m^{n-2} + (m-1)\Big[\dots\Big[m^3 + (m-1)\Big[m^2 + (m-1)\Big[1\Big]\Big]\Big]\dots\Big]\Big] \end{aligned}$$

Algorithm 4.4 *generalAlgorithm*: The general algorithm for fictitious play or rational learning predators to decide which prey to chase.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $P$: A probability matrix, with $P_{ij}$ denoting the probability predator $i$ selects prey $j$.
Require: $a_{prev}$: The previously selected prey, if any.
Require: $E[a_{prev}]$: The expected utility for the previous prey at this stage, and 0 if no prey
    selected.
1: $a^* \leftarrow a_{prev}$
2: $E[a^*] \leftarrow E[a_{prev}]$
3: for $a_i \in \{0, \ldots, m-1\}$ do
4:   $p^*(a_i) \leftarrow 0$
5:   for $a_{-i} \in \{0, \ldots, m^n - 1\}$ do
6:     $p^*(a_i, a_{-i}) \leftarrow 1$
7:     for $a' \in \{0, \ldots n-1\}$ do
8:       $\hat{a} \leftarrow resolveAction(n, m, a_{-i}, a')$
9:       if $\hat{a} = a_i$ then
10:        $p^*(a_i, a_{-i}) \leftarrow p^*(a_i, a_{-i}) * 0$
11:      else
12:        $p^*(a_i, a_{-i}) \leftarrow p^*(a_i, a_{-i}) * P_{a', \hat{a}}$
13:      end if
14:    end for
15:    $p^*(a_i) \leftarrow p^*(a_i) + p^*(a_i, a_{-i})$
16:  end for
17:  $E[a_i] \leftarrow u_i(a_i) * p^*(a_i)$
18:  if $E[a_i] > E[a^*]$ then
19:    $a^* \leftarrow a_i$
20:    $E[a^*] \leftarrow E[a_i]$
21:  end if
22: end for
23: $a_{prev} \leftarrow a^*$
24: $E[a_{prev}] \leftarrow E[a^*]$
25: return $a^*$
```

$$= (m-1)^0 m^{n-1} + (m-1)^1 m^{n-2} + (m-1)^2 m^{n-3} + \ldots + (m-1)^{n-4} m^3$$

$$+ (m-1)^{n-3} m^2 + (m-1)^{n-2} m^1 + (m-1)^{n-1} m^0$$

$$= \frac{\sum_{i=1}^{n-1} (m-1)^i m^{n-i}}{m-1} + (m-1)^{n-1} = \frac{m^n \left[\frac{1-(\frac{m-1}{m})^n}{1-(\frac{m-1}{m})}\right] - m^n}{m-1} + (m-1)^{n-1}$$

$$= \left[\frac{m^n - m^n(\frac{m-1}{m})^n}{(m-1) - (\frac{m-1}{m})(m-1)}\right] - \frac{m^n}{m-1} = \left[\frac{m^n - (m-1)^n}{(m-1) - \frac{(m-1)^3}{m}}\right] - \left[\frac{m^n - (m-1)^n}{(m-1)}\right]$$

$$= [m^n - (m-1)^n]\left[\frac{\frac{(m-1)^2}{m}}{(m-1)^2 - \frac{(m-1)^2}{m}}\right] = [m^n - (m-1)^n]\left[\frac{1}{m - (m-1)}\right]$$

$$z = m^n - (m-1)^n \tag{4.11}$$

Therefore, the number of zeros is simply $z = m^n - (m-1)^n$. This number is equivalent to the

Algorithm 4.5 *zeroCount*: A recursive function to count the number of zeros based on n and m.

```
Require: $n$: The number of predators.
Require: $m$: The number of prey.
1: if $n = 1$ then
2:     return 1
3: else
4:     return $m^{n-1} + (m-1) * zeroCount(n-1, m)$
5: end if
```

number of wasted steps during computation, since the zero is a result of m multiplications, one of which is zero. The number is quite large, especially when n and m are very different. Since the total number of computations in the summation is m^n, the number of ones is $o = m^n - z$. To see the full range of zero values, we plotted the percentage of zeros ($\frac{z}{m^n}$) as a heat map in figure 4.4.

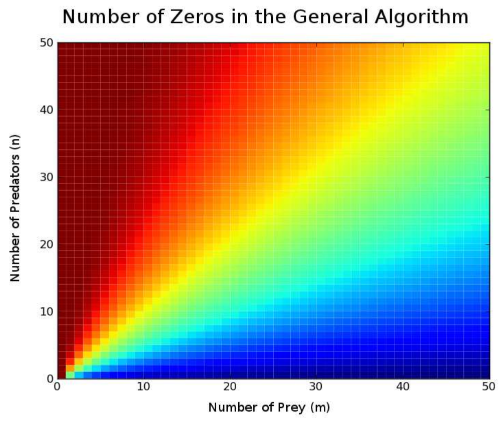


Figure 4.4: A heat map demonstrating the percentage of zeros in a single predator's decision-making process. The red area is close to 100%, the yellow area is around 50%, and the blue area is around 0%. The x-axis and y-axis represent n and m, respectively.

In summary, the number of zeros grows rapidly with the number of observed predators, and drops off rapidly with the number of prey. Intuitively this makes perfect sense, since we have determined in section 4.6.2 that the $p_i(a^*)$ variable is the probability no other predator is chasing prey a_i. The more predators that are observed, the lower this probability becomes. This probability uses both the $u_i(a_i)$ summation and 0 summation, meaning the 0 summation dominates here. The next section describes the way we leverage this concept to improve run-time performance.

4.6.4 Optimized Algorithm

Upon further inspection of the equations, one finds that there is a lot of wasted computation. Ideally, we don't want to compute the probability if the action $a_i \in a_{-i}$, since it will just be multiplied by a zero. Note that we cannot use simple memory management tricks with sets since the worst-case computation still requires every action to be checked during every step in the summation. Our goal is to skip over the cases for which $a_i \in a_{-i}$ through the highly structured manner in which they are computed.

This is not enough to improve performance for every case because the number of zeros is dominated by the proportion of predators to prey. We begin our optimization by reexamining the equations in section 4.6.2. From these, we may write the following equivalent statements.

$$\begin{aligned}
a_i^* &= \underset{a_i \in A_i}{\arg\max} \Big[u_i(a_i) p_i^*(a_i) \Big] \\
&= \underset{a_i \in A_i}{\arg\max} \Big[u_i(a_i) \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \notin a_{-i}}} \prod_{a' \in a_{-i}} p_i(a') + 0 \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \in a_{-i}}} \prod_{a' \in a_{-i}} p_i(a') \Big] \\
&= \underset{a_i \in A_i}{\arg\max} \Big[u_i(a_i) \Big(1 - \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \in a_{-i}}} \prod_{a' \in a_{-i}} p_i(a') \Big) + 0 \Big(1 - \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \notin a_{-i}}} \prod_{a' \in a_{-i}} p_i(a') \Big) \Big]
\end{aligned}$$

Similar to our notation for $p_i^*(a_i)$, we let $p_i^{**}(a_i)$ be the following.

$$p_i^{**}(a_i) = \sum_{\substack{a_{-i} \in A_{-i} \\ a_i \in a_{-i}}} \prod_{a' \in a_{-i}} p_i(a')$$

$$1 = p_i^*(a_i) + p_i^{**}(a_i)$$

Finally, we are able to rewrite this in the form below.

$$\begin{aligned}
a_i^* &= \underset{a_i \in A_i}{\arg\max} \Big[u_i(a_i)(1 - p_i^{**}(a_i)) + 0(1 - p_i^*(a_i)) \Big] \\
&= \underset{a_i \in A_i}{\arg\max} \Big[u_i(a_i)(1 - p_i^{**}(a_i)) \Big]
\end{aligned}$$

In other words, we are able to compute one minus the probability of the "0" side. This enables a simple substitution based on the quantity of zeros that would have been multiplied by these probabilities. The end result of this process is that we will be able to compute $p_i^*(a_i)$ if the majority of zero-multiplications is greater for $p_i^{**}(a_i)$, and conversely compute $p_i^{**}(a_i)$ if the majority of zero-multiplications is greater for $p_i^*(a_i)$. The final form of this algorithm is given in algorithms 4.6, 4.7, and 4.8.

This new algorithm improves upon the general algorithm's computation time by breaking it into two parts and taking the minimum of the two. Recall that the total number of permutations is given by m^n. Therefore the number of *non*zero terms is given by $m^n - (m^n - (m-1)^n) = (m-1)^n$. Ideally, this would yield the complexity given in equation 4.12; however, in the actual

Algorithm 4.6 *optimizedAlgorithm*: The optimized algorithm for fictitious play or rational learning predators to decide which prey to chase.

```
Require: $n$: The number of predators.
Require: $m$: The number of prey.
1: // If the percentage of zeros is greater than $\frac{1}{2}$, then perform component 1; component 2 otherwise.
2: if $m^n - (m-1)^n \geq \frac{m^n}{2}$ then
3:   return optimizedAlgorithmComponent1()
4: else
5:   return optimizedAlgorithmComponent2()
6: end if
```

Algorithm 4.7 *optimizedAlgorithmComponent1*: The first component for the optimized algorithm for fictitious play or rational learning predators to decide which prey to chase.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $P$: A probability matrix, with $P_{ij}$ denoting the probability predator $i$ selects prey $j$.
Require: $a_{prev}$: The previously selected prey, if any.
Require: $E[a_{prev}]$: The expected utility for the previous prey at this stage, and 0 if no prey selected.
1: $a^* \leftarrow a_{prev}$
2: $E[a^*] \leftarrow E[a_{prev}]$
3: for $a_i \in \{0, \dots, m-1\}$ do
4:   $p^*(a_i) \leftarrow 0$
5:   for $a_{-i} \in \{0, \dots, (m-1)^n - 1\}$ do
6:     $p^*(a_i, a_{-i}) \leftarrow 1$
7:     for $a' \in \{0, \dots n-1\}$ do
8:       $\hat{a} \leftarrow resolveAction(n, m-1, a_{-i}, a')$
9:       if $\hat{a} \geq a_i$ then
10:        $\hat{a} \leftarrow \hat{a} + 1$
11:      end if
12:      $p^*(a_i, a_{-i}) \leftarrow p^*(a_i, a_{-i}) * P_{a',\hat{a}}$
13:    end for
14:    $p^*(a_i) \leftarrow p^*(a_i) + p^*(a_i, a_{-i})$
15:  end for
16:  $E[a_i] \leftarrow u_i(a_i) * p^*(a_i)$
17:  if $E[a_i] > E[a^*]$ then
18:    $a^* \leftarrow a_i$
19:    $E[a^*] \leftarrow E[a_i]$
20:  end if
21: end for
22: $a_{prev} \leftarrow a^*$
23: $E[a_{prev}] \leftarrow E[a^*]$
24: return $a^*$
```

Algorithm 4.8 *optimizedAlgorithmComponent2*: The second component for the optimized algorithm for fictitious play or rational learning predators to decide which prey to chase.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $P$: A probability matrix, with $P_{ij}$ denoting the probability predator $i$ selects prey $j$.
Require: $a_{prev}$: The previously selected prey, if any.
Require: $E[a_{prev}]$: The expected utility for the previous prey at this stage, and 0 if no prey
    selected.
 1: $a^* \leftarrow a_{prev}$
 2: $E[a^*] \leftarrow E[a_{prev}]$
 3: for $a_i \in \{0, \dots, m-1\}$ do
 4:   $p^{**}(a_i) \leftarrow 0$
 5:   for $\alpha \in \{0, \dots, n-1\}$ do
 6:     for $\beta \in \{0, \dots, (m-1)^{\alpha} - 1\}$ do
 7:       for $\gamma \in \{0, \dots, m^{n-\alpha-1} - 1\}$ do
 8:         $a_{-i} \leftarrow 0$
 9:         for $a' \in \{0, \dots, n-1\}$ do
10:           $\hat{a} \leftarrow 0$
11:           if $a' = \alpha$ then
12:             $\hat{a} \leftarrow a_i$
13:           else if $a' < \alpha$ then
14:             $\hat{a} \leftarrow resolveAction(n, m-1, \beta, a')$
15:           else if $a' > \alpha$ then
16:             $\hat{a} \leftarrow resolveAction(n, m, \gamma, a' - \alpha - 1)$
17:           end if
18:           $a_{-i} \leftarrow a_{-i} + m^{a'} * \hat{a}$
19:         end for
20:         $p^{**}(a_i, a_{-i}) \leftarrow 1$
21:         for $a' \in \{0, \dots n-1\}$ do
22:           $\hat{a} \leftarrow resolveAction(n, m-1, a_{-i}, a')$
23:           $p^{**}(a_i, a_{-i}) \leftarrow p^{**}(a_i, a_{-i}) * P_{a',\hat{a}}$
24:         end for
25:         $p^{**}(a_i) \leftarrow p^{**}(a_i) + p^{**}(a_i, a_{-i})$
26:       end for
27:     end for
28:   end for
29:   $E[a_i] \leftarrow u_i(a_i) * (1 - p^{**}(a_i))$
30:   if $E[a_i] > E[a^*]$ then
31:     $a^* \leftarrow a_i$
32:     $E[a^*] \leftarrow E[a_i]$
33:   end if
34: end for
35: $a_{prev} \leftarrow a^*$
36: $E[a_{prev}] \leftarrow E[a^*]$
37: return $a^*$
```

implementation it is slightly more complicated.

$$O(nm\min\{m^n-(m-1)^n,(m-1)^n\}) \tag{4.12}$$

Algorithm 4.7 takes the worst-case order as given by equation 4.13 because it essentially removes one prey from the computation and performs a simple adjustment mechanism instead. To better understand the algorithm, we will again turn to the example given in table 4.2. In this example, we have fixed the prey to be A and marked a "Yes" if prey A is also selected by one of the other three predators for each row in the permutations. The marked rows may therefore be skipped during the computation of expected utility, which is equivalent to skipping over the prey A completely. The algorithm 4.7 implements this behavior, using a simple offset to correct the index during the control loops.

$$O(n(m-1)^{n+1}) \tag{4.13}$$

Algorithm 4.8 is a bit more complicated. The overarching idea is to skip over the nonzero terms. Referring back to the example in table 4.2, it would attempt to skip over the ones labeled with a "No." The complexity analysis examines the control loops once again, following the logic below that produces the complexity in equation 4.14.

$$\begin{aligned}
mn(\sum_{\alpha=1}^{n}[(m-1)^{\alpha}m^{n-\alpha-1}n]) &= m^{n+1}n^2\Big(\sum_{\alpha=1}^{n}\frac{(m-1)^{\alpha}}{m^{\alpha+1}}\Big)\\
&= \frac{m^{n+1}}{m}n^2\Big(\sum_{\alpha=1}^{n}\Big[\frac{(m-1)}{m}\Big]^{\alpha}\Big) \qquad \text{a geometric series}\\
&= m^n n^2\Big(\frac{1-\frac{(m-1)^{n-1}}{m^{n-1}}}{1-\frac{(m-1)}{m}}\Big)\\
&= \frac{m^n n^2-\frac{m^n n^2(m-1)^{n-1}}{m^{n-1}}}{1-\frac{(m-1)}{m}}\\
&= \frac{m^n n^2}{\frac{1}{m}}-\frac{mn^2(m-1)^{n-1}}{\frac{1}{m}}\\
&= m^{n+1}n^2-m^2n^2(m-1)^{n-1}
\end{aligned}$$

$$O(m^2n^2[m^{n-1}-(m-1)^{n-1}]) \tag{4.14}$$

4.6.5 Optimization Experimentation

With the optimized algorithm (4.6, 4.7, and 4.8) in place, we conducted a small experiment to ensure its performance improvement over the general algorithm (4.4). First the experiment generated a random probability matrix P, as described in the algorithms, in addition to random payoff values $u_i(a_i)$ for all prey $a_i \in A_i$. Next it ran the general and optimized algorithms in sequence. This process was repeated for all permutations of 2 to 6 predators and prey, a total of 25 configurations. For each of these configurations we ran 100 trials. We also used

Computation Time - General Algorithm (microseconds)

	2	3	4	5	6
2	0.00007895	0.00023934	0.00050973	0.00086932	0.00145235
3	0.00019795	0.00075196	0.00241358	0.00581225	0.01177954
4	0.00047378	0.00289942	0.01152901	0.03566806	0.08918241
5	0.00092759	0.00995283	0.05683612	0.22131951	0.66330532
6	0.00210143	0.03544956	0.27188195	1.29954453	4.69254692

Computation Time - Optimized Algorithm (microseconds)

	2	3	4	5	6
2	0.00004506	0.00012087	0.00043205	0.00062894	0.00088896
3	0.00005106	0.00029366	0.00107869	0.00575608	0.01003664
4	0.00005649	0.00068619	0.00400882	0.01553964	0.04632215
5	0.00006166	0.00145747	0.01449137	0.07647226	0.28594011
6	0.00006712	0.00340148	0.05143805	0.36563757	1.67794588

Computation Time - Reduction in Computation (percentage)

	2	3	4	5	6
2	42.92%	49.49%	15.23%	27.65%	38.79%
3	74.20%	60.94%	55.30%	00.96%	14.79%
4	88.07%	76.33%	65.22%	56.43%	48.05%
5	93.35%	85.35%	74.50%	65.44%	56.89%
6	96.80%	90.40%	81.08%	71.86%	64.24%

Table 4.3: Results for the computation time of the general algorithm (above), the optimized algorithm (middle), and the reduction in computation ratio (below) The number of predators n varies over the rows, and the number of prey m varies over the columns. Each cell represents 100 trials.

two kinds of measurements: computation time in microseconds (i.e. 1×10^{-6} seconds) and a counter recording the number of times the algorithm entered a control loop. The former provides a realistic performance estimate but might be subject to hardware and implementation specific biases. The latter obviates the issue by counting the number of times a control loop is entered, which is the dominating factor in determining algorithmic performance.

The results from our optimization experimentation are given in tables 4.3 and 4.5. All trials were performed on a 3.2 GHz Intel(R) Pentium(R) 4 CPU with 1 GB of RAM under OpenSuse Linux 11.4. In both tables, let A denote the general algorithm subtable B denote the optimized algorithm subtable. Let the cells A_{ij}, B_{ij}, and C_{ij} be from the general algorithm table, optimized algorithm table, and performance improvement table, respectively, with $i, j \in \{2, 6\}$ representing row i, and column j. The reduction in computation percentage, a measure of performance improvement, is given by equation 4.15 below.

$$C_{ij} = 1 - \frac{A_{ij}}{B_{ij}} \tag{4.15}$$

To gain a better understanding of the results, we have included the theoretical reduction in computation in table 4.4. This was taken from the ratio of zero multiplications over the total multiplications, and is graphically represented in figure 4.5. Our theoretical reduction rate closely

Theoretical Reduction in Computation (percentage)

	2	3	4	5	6
2	75.00%	55.55%	56.25%	64.00%	69.45%
3	87.50%	70.37%	57.81%	51.20%	57.88%
4	93.75%	80.24%	68.35%	59.04%	51.77%
5	96.87%	86.83%	76.26%	67.23%	59.81%
6	98.43%	91.22%	82.20%	73.78%	66.51%

Table 4.4: The ideal reduction in computation from the theoretical analysis of the number of zeros. The number of predators n varies over the rows, and the number of prey m varies over the columns. Each cell was computed by taking the maximum of the ratio of zero or one multiplications to total multiplications, e.g. for the zero case, $\frac{m^n-(m-1)^n}{m^n} = 1 - (\frac{(m-1)}{m})^n$.

Operation Count - General Algorithm (operations)

	2	3	4	5	6
2	26	84	196	380	654
3	66	327	1028	2505	5190
4	162	1218	5124	15630	38886
5	386	4377	24580	93755	279942
6	898	15312	114692	546880	1959558

Operation Count - Optimized Algorithm (operations)

	2	3	4	5	6
2	8	39	112	175	252
3	10	99	436	1345	2394
4	12	243	1624	6405	18756
5	14	579	5836	30725	112506
6	16	1347	20416	143365	656256

Operation Count - Reduction in Computation (percentage)

	2	3	4	5	6
2	69.23%	53.57%	42.85%	53.94%	61.46%
3	84.84%	69.72%	57.58%	46.30%	53.87%
4	92.59%	80.04%	68.30%	59.02%	51.76%
5	96.37%	86.77%	76.25%	67.22%	59.81%
6	98.21%	91.20%	82.19%	73.78%	66.50%

Table 4.5: Results for the control loop operation count of the general algorithm (above), the optimized algorithm (middle), and the reduction in computation ratio (below). The number of predators n varies over the rows, and the number of prey m varies over the columns. Each cell represents 100 trials.

matches the actual reduction rate in both metrics. There are however some discrepancies because the key assumption we made was that the proportion of zeros to non-zeros directly maps to a performance improvement; this is not always the case. Aside from measurement inaccuracies, the additional overhead for the optimized algorithm is the most likely culprit. In any case, it a substantial improvement in performance over the original, often many orders of magnitude faster. Lastly, we compared the actual expected utilities of the general algorithm versus the returned expected utilities of the optimized algorithm, and found that they matched exactly to within machine precision.

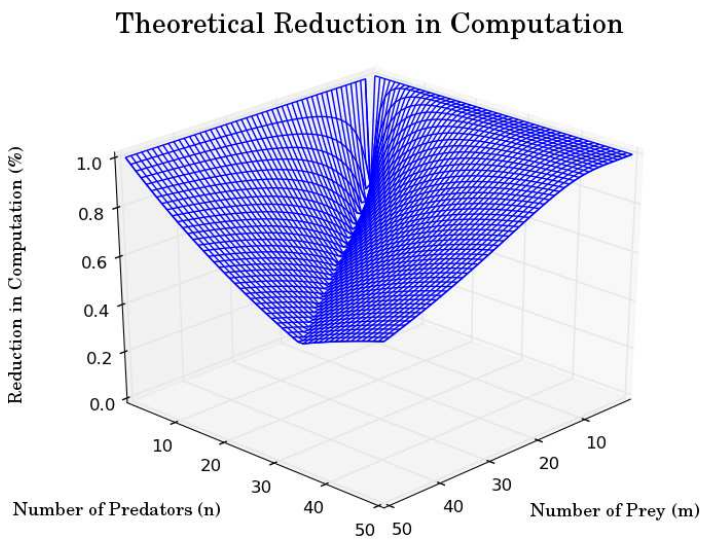


Figure 4.5: The theoretical reduction in computation, visually elaborating on table 4.4.

4.7 Experimentation

We simulated Defender with N, $M \in \{1, \ldots, 8\}$, $|N| = |M|$, for each of the five predator learning algorithms and two types of prey. This provides a total of *80* different combinations. For all of our experiments, we used the environmental parameters $E = \{s^*, c^*, t^*, \phi^*,\ (\delta_{loc}, \delta_{vel}, \delta_{\phi}),\ (p_{miss}, p_{err}),\ (a_{vmax}, a_{tmax}, a_{update})\} = \{$ 100 units, 3.3 units, 10 seconds, $\frac{\pi}{3}$, (1 unit, 3 units, 0.01 units), (5%, 5%), (0.3 units, 0.75 units, $\frac{1}{2}$ second) $\}$. We do provide some experimentation with these values adjusted. For each particular configuration of predators, prey, and agent quantity, we have run 5000 trials to obtain an accurate result of the initialization and simulation space. The simulation environment itself was written in C++, using OpenGL for visual output. All trials were performed on a 3.2 GHz Intel(R) Pentium(R) 4 CPU with 1 GB of RAM under OpenSuse Linux 11.4.

4.7.1 Overall Results

We used two different metrics for comparison. Our main goal was to compare the percentage of prey that were successfully captured by the predators. This metric clearly delineates which algorithm performs the best overall. It has also been used in the predator-prey and multi-agent literature. For example, Gmytrasiewicz and Durfee [29]'s air defense simulations used the number of prey captured, which is proportional to the percentage captured. Our second metric was the percentage of trials in which the predators managed to capture all prey. This kind of measurement

	1	2	3	4	5	6	7	8
Simple								
Fictitious Play	97.02%	95.79%	93.99%	92.28%	91.96%	90.74%	90.29%	90.17%
Rational Learning	98.10%	93.78%	91.27%	89.15%	88.55%	88.06%	87.80%	87.55%
Regret Matching	98.22%	93.32%	88.56%	85.10%	82.99%	81.79%	80.60%	80.14%
Minimax Regret	97.60%	57.85%	50.39%	47.28%	n/a	n/a	n/a	n/a
Greedy	97.20%	87.80%	83.25%	82.57%	81.25%	81.20%	81.07%	81.28%
Maneuvering								
Fictitious Play	96.98%	93.62%	92.23%	90.19%	88.92%	88.41%	87.71%	87.41%
Rational Learning	95.96%	91.83%	88.61%	87.03%	85.91%	85.31%	85.25%	85.10%
Regret Matching	97.74%	90.78%	85.80%	82.67%	80.06%	78.79%	78.06%	77.31%
Minimax Regret	95.32%	66.09%	56.38%	52.84%	n/a	n/a	n/a	n/a
Greedy	95.10%	87.48%	82.93%	80.93%	81.12%	80.64%	80.36%	80.52%

Table 4.6: Results for the percentage of prey captured in configurations of {simple, maneuvering} prey vs. {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 8 agents. These results summarize 5000 trials for each cell.

	1	2	3	4	5	6	7	8
Simple								
Fictitious Play	97.02%	97.72%	82.40%	71.80%	64.26%	54.06%	45.22%	39.64%
Rational Learning	98.10%	87.64%	74.74%	60.02%	50.50%	41.10%	33.72%	27.42%
Regret Matching	98.22%	86.84%	68.36%	50.62%	36.50%	26.38%	18.22%	12.94%
Minimax Regret	97.60%	35.70%	11.16%	04.74%	n/a	n/a	n/a	n/a
Greedy	97.20%	75.68%	51.54%	39.50%	26.46%	19.36%	14.42%	11.00%
Maneuvering								
Fictitious Play	96.98%	87.48%	78.14%	65.02%	53.84%	45.62%	36.60%	30.26%
Rational Learning	95.96%	83.96%	98.26%	54.48%	42.42%	33.58%	26.52%	21.34%
Regret Matching	97.74%	82.42%	61.98%	44.76%	30.12%	19.44%	14.34%	08.84%
Minimax Regret	95.32%	42.16%	15.80%	06.78%	n/a	n/a	n/a	n/a
Greedy	95.10%	74.96%	51.20%	34.46%	25.68%	18.84%	13.30%	10.02%

Table 4.7: Complete capture percentage results for configurations of {simple, maneuvering} prey vs. {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 8 agents. These results summarize 5000 trials for each cell.

aligns itself with competitive sports. It also follows from the original predator-prey problem, and of course robotic simulations like the RoboCup. Finally, we will briefly discuss the average final location of all agents on the field, and how it relates to each configuration.

Tables 4.6 and 4.7 describe the results from our simulations. We grouped the trials by the type of prey to more easily compare the various learning algorithms. We let the term *number of agents* below refer to the quantity of agents on a specific team, i.e. $|N|$ or equivalently $|M|$. Lastly, minimax regret became too computationally infeasible to run 5000 trials beyond 4 agents and thus have omitted the results.

We also provide graphs that coincide with tables 4.6 and 4.7. We again have partitioned the data over the type of prey. These graphs are found in figures 4.6, 4.7, 4.8, and 4.9.

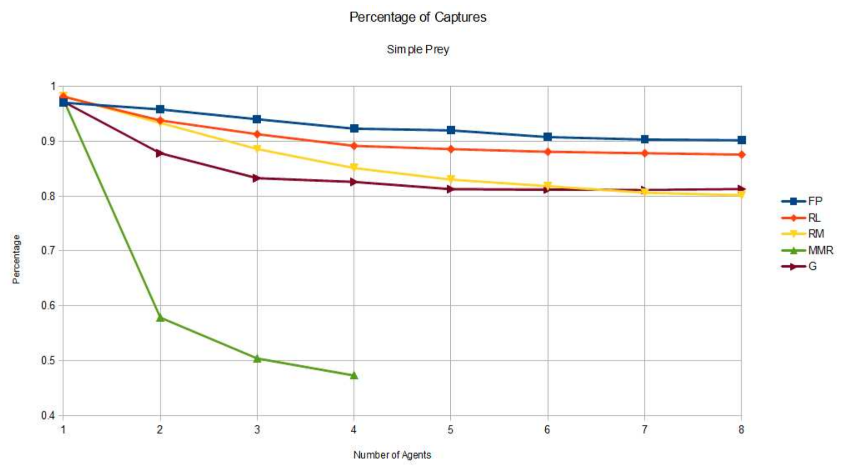


Figure 4.6: The percentage of simple prey captured. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8.

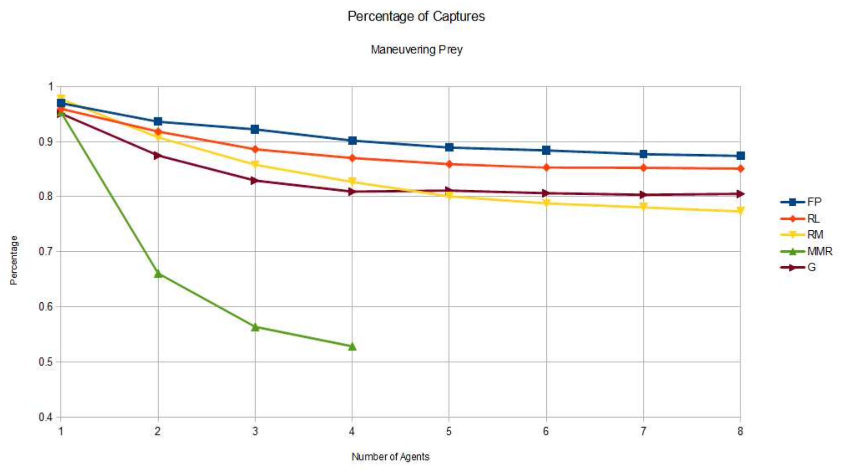


Figure 4.7: The percentage of maneuvering prey captured. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8.

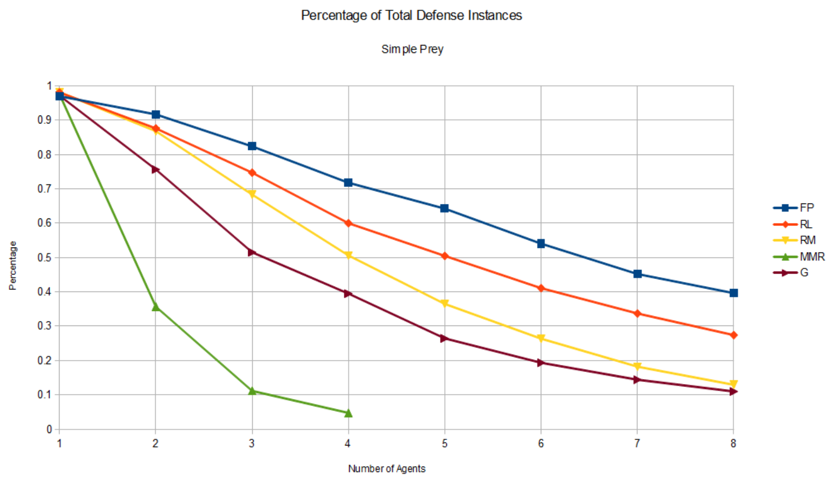


Figure 4.8: The complete capture percentage against simple prey. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8.

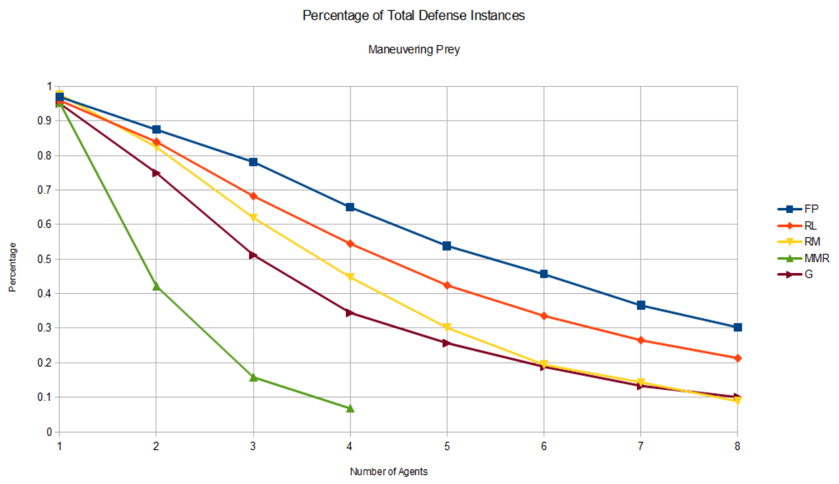


Figure 4.9: The complete capture percentage against maneuvering prey. This compares fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 8.

As another point of comparison, we present the average final location of all agents for each of the five learning algorithms from the simulations. This perspective gives a good idea of the rapid convergence to cooperation, or divergence as the case may be. These results are presented in figure 4.10.

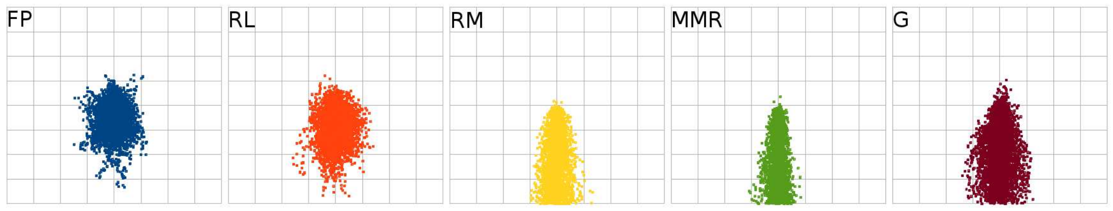


Figure 4.10: This figure shows the average final location of the agents for each of the 5 algorithms. From left to right: fictitious play, rational learning, regret matching, minimax regret, and greedy. This describes four predators against four maneuvering prey.

4.7.2 Selected Examples

In an effort to underline the major properties of the algorithms, we present three examples. For the sake of brevity, we will only present fictitious play, regret matching, and greedy agents. Fictitious play and rational learning agents behave in a similar manner, as do regret matching and minimax regret agents. We selected $n = m = 4$ for our demonstration because it provides a balance of complexity and understandability. Each example was chosen to specifically capture what we generally observe to be the emergent behaviors. Figures 4.11, 4.12, and 4.13 present a specific case of fictitious play, regret matching, and the simple greedy strategy, respectively. These figures will be discussed in section 4.8.

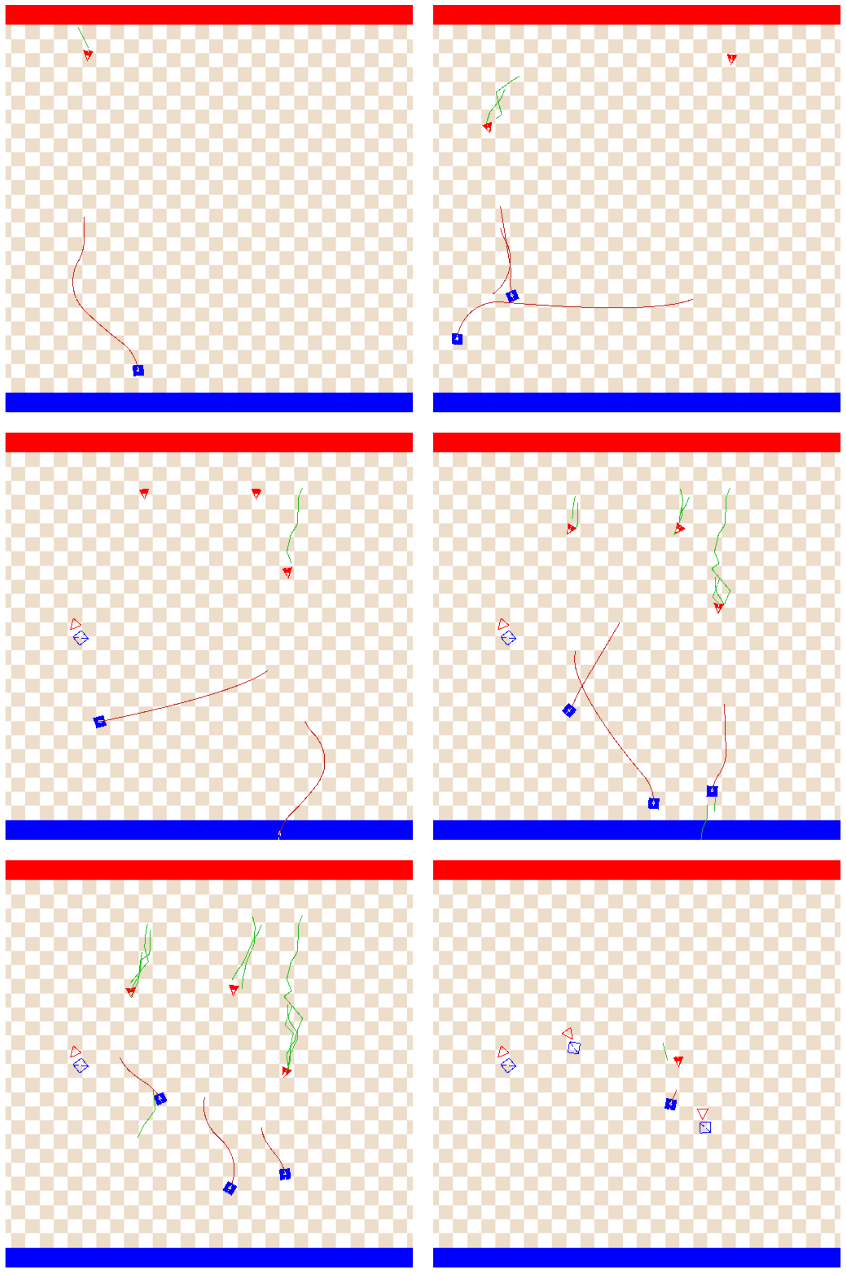


Figure 4.11: An extended example of fictitious play predators chasing maneuvering prey ($n = m = 4$). The time sequence follows: top left, top right, middle left, middle right, bottom left, bottom right. Rational learning predators also show this kind of behavior.

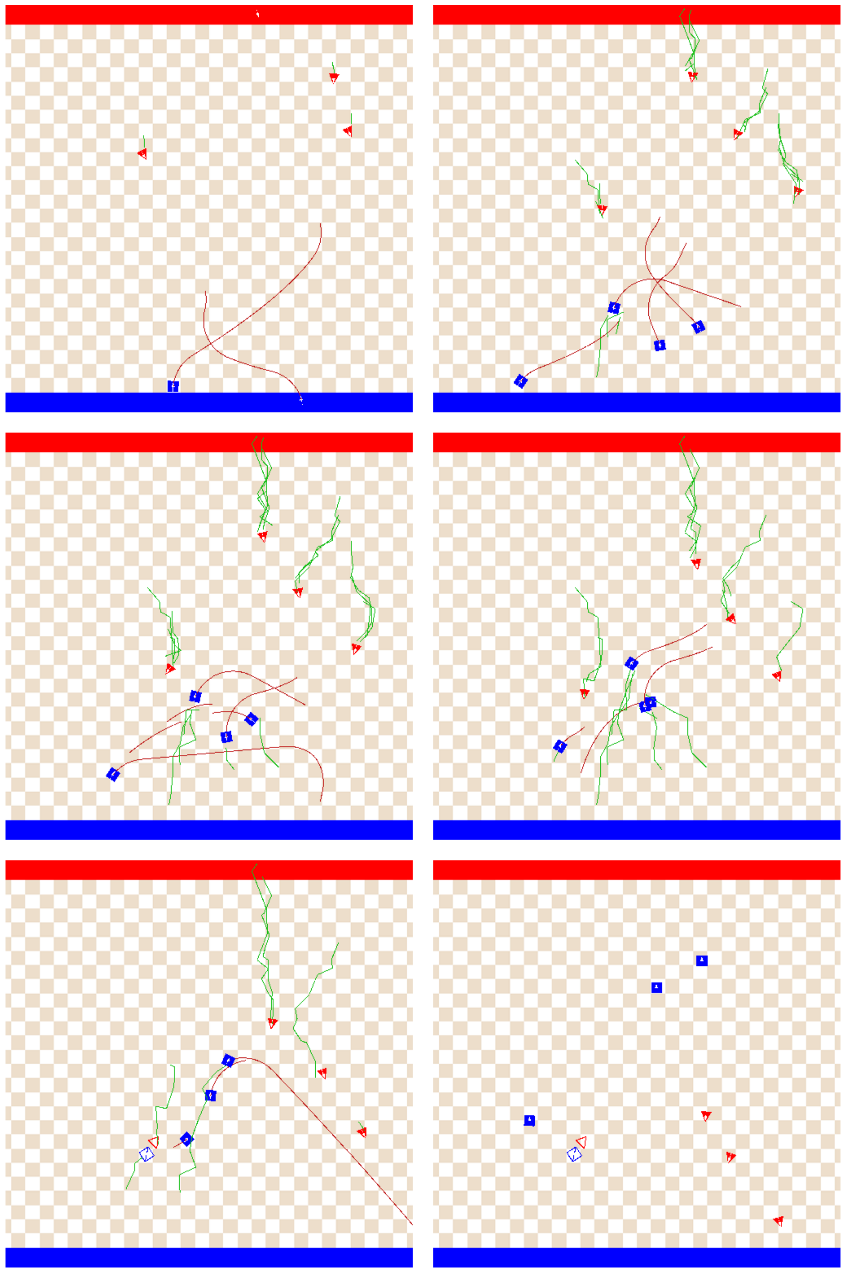


Figure 4.12: An extended example of regret matching predators chasing maneuvering prey ($n = m = 4$). The time sequence follows: top left, top right, middle left, middle right, bottom left, bottom right. Minimax regret predators also show this kind of behavior.

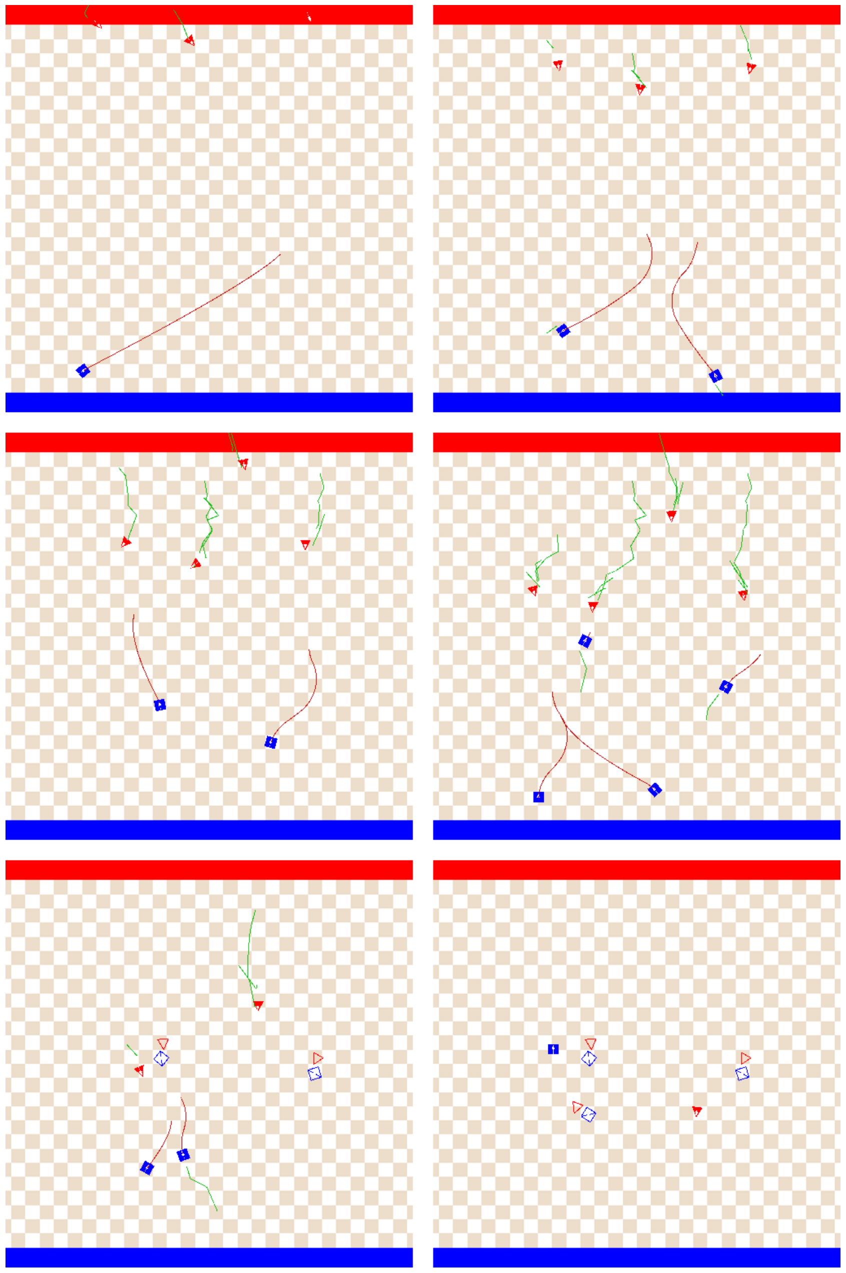


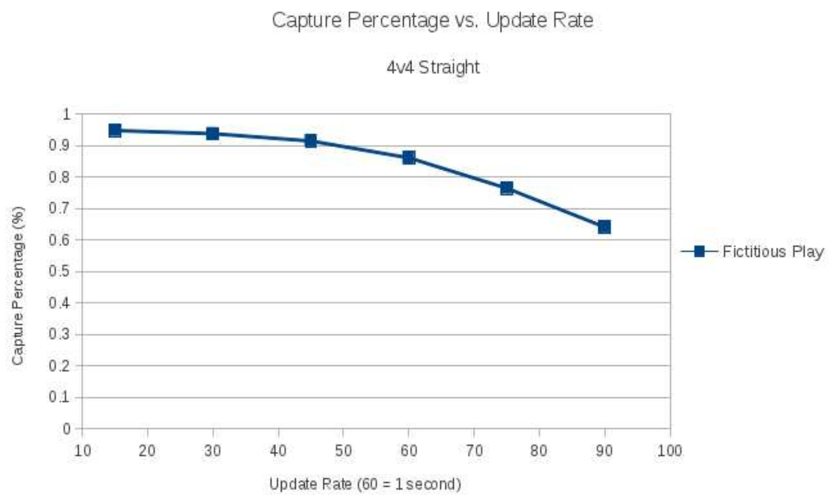
Figure 4.13: An extended example of greedy predators chasing maneuvering prey ($n = m = 4$). The time sequence follows: top left, top right, middle left, middle right, bottom left, bottom right.

4.7.3 Modifying Environmental Parameters

The space of all possible combinations of environmental parameters is quite large, in addition to the already massive dimension of agent learning algorithms and predator/prey quantities.

Informally, we observed the same general behavior of each learning algorithm, including the successful cooperation of action-based fictitious play agents in this dynamic anti-coordination game. We tested various field (area) sizes, speed differences, etc., and they all yielded highly predictable results. We would like to explore one facet of the environment directly: the time constraints of cooperative convergence.

We conducted a small experiment to understand how update rate affects the predators. Primarily, we sought to determine the *rate of performance decrease* as a function of the update rate. We concentrated on the standard $n = m = 4$ case as it retains the reliable balance stated in the previous subsection and is a very common agent "team size" in the literature [5, 47, 29, 28, 16, 55]. Figure 4.14 shows the update rate's effect on performance in 300 trials for each configuration with fictitious play predators and straight moving prey.



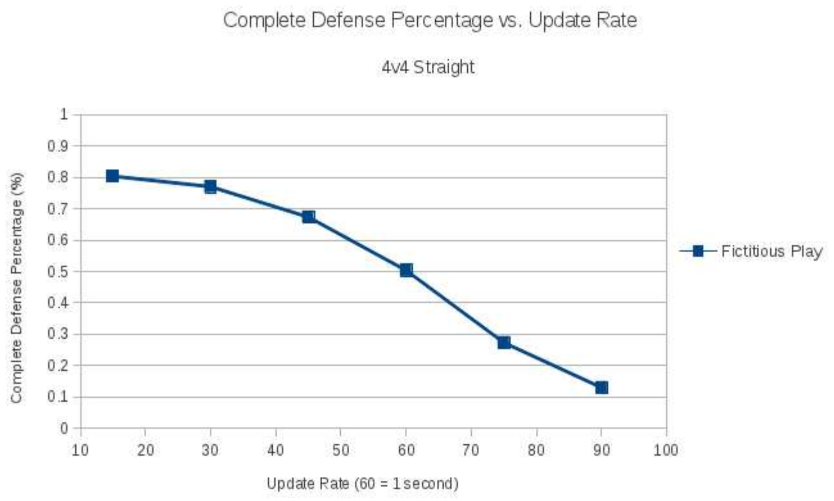


Figure 4.14: A summary of the effect that update rate has on the performance of four fictitious play predators chasing four straight moving prey. Each point represents the mean of 300 trials.

Ad Hoc Teams - Fictitious Play

	2	3	4	5	6	7	8
Rational Learning	92.4%	91.9%	91.6%	90.6%	90.5%	91.7%	90.8%
Regret Matching	86.2%	86.7%	87.3%	87.6%	88.1%	89.9%	89.5%
Minimax Regret	71.6%	79.1%	82.3%	83.9%	86.4%	n/a	n/a
Greedy	90.2%	91.3%	88.8%	89.8%	89.0%	90.9%	89.6%
Overall	85.1%	86.7%	87.3%	87.6%	88.1%	89.9%	89.5%

Table 4.8: Results from ad hoc teams with fictitious play predators and one perturbed predator given by the row. Team sizes are given by each column. Each cell represents the mean over 250 trials. The final row states the cumulative results from that column, a total of 1000 trials per cell.

Ad Hoc Teams - Rational Learning

	2	3	4	5	6	7	8
Fictitious Play	91.0%	91.9%	89.3%	88.8%	88.2%	87.8%	88.7%
Regret Matching	85.0%	82.3%	82.3%	83.2%	85.6%	86.6%	86.7%
Minimax Regret	66.8%	77.3%	81.5%	82.1%	84.4%	n/a	n/a
Greedy	89.0%	86.9%	88.4%	87.1%	86.1%	87.3%	87.7%
Overall	82.9%	84.6%	85.3%	85.3%	86.2%	87.2%	87.7%

Table 4.9: Results from ad hoc teams with rational learning predators and one perturbed predator given by the row. Team sizes are given by each column. Each cell represents the mean over 250 trials. The final row states the cumulative results from that column, a total of 1000 trials per cell.

4.7.4 Ad Hoc Teams

As is evident by sections 4.6 and 4.7, our focus is on the successful fictitious play and rational learning predators as a solution to Defender and other highly dynamic role assignment problems. With this in mind, we now turn our attention to the state of the art for multi-agent systems. As discussed in chapter 2, many researchers are experimenting with ad hoc teams [5, 81, 82, 97, 90]. In this subsection, we experiment with another kind of ad hoc team setting in which we perturbed one predator in a team of fictitious play or rational learning predators. We then recorded the subsequent effects on performance and cooperative convergence. This experiment also tests which algorithm is more robust to heterogeneous teammates not known *a priori*.

We ran our experiments for 2 to 8 agents with $n = m$. To get an idea of the robustness of fictitious play versus rational learning, we perturbed one agent at random on the team of predators. This agent was selected from the set of the other four learning algorithms, e.g. for fictitious play the perturbed predator could be rational learning, regret matching, minimax regret, or greedy. We ensured that over all trials, each other algorithm was selected an equal proportion of times. There were 1000 trials in total and thus 250 trials per learning algorithm for the perturbed agent. Tables 4.8 and 4.9 numerically summarize our results; figure 4.15 provides a visual representation as well. Figures 4.16 and 4.17 present the distinct results from fictitious play and rational learning teams, respectively.

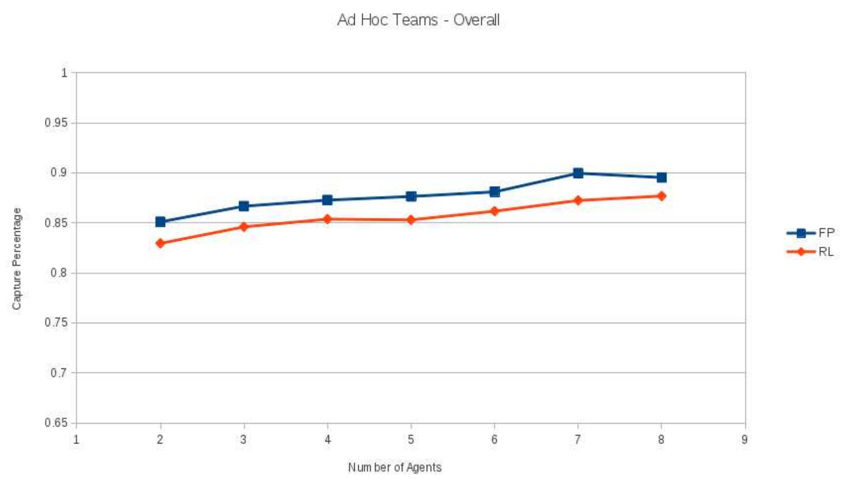


Figure 4.15: A visual representation of the overall results for ad hoc teams of fictitious play and rational learning predators from tables 4.8 and 4.9. Each data point on each line summarizes 1000 trials of that particular configuration.

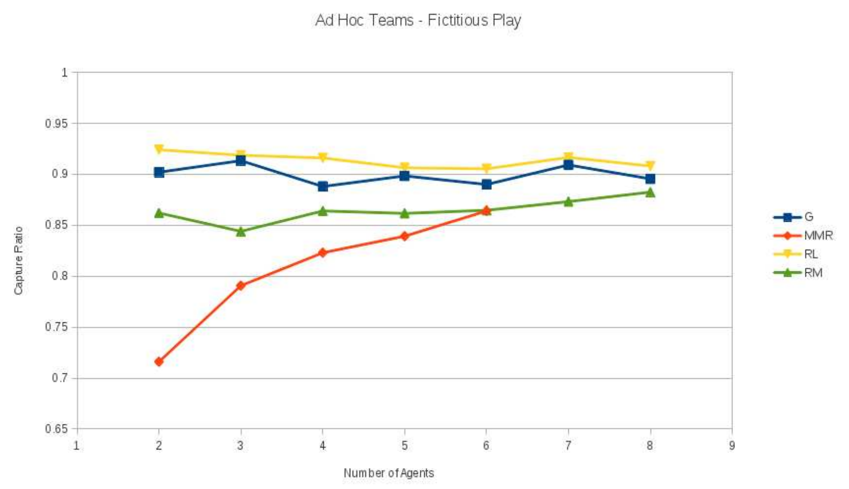


Figure 4.16: A visual representation of the individual results for ad hoc teams of fictitious play predators from table 4.8. Each data point on each line summarizes 250 trials of that particular configuration.

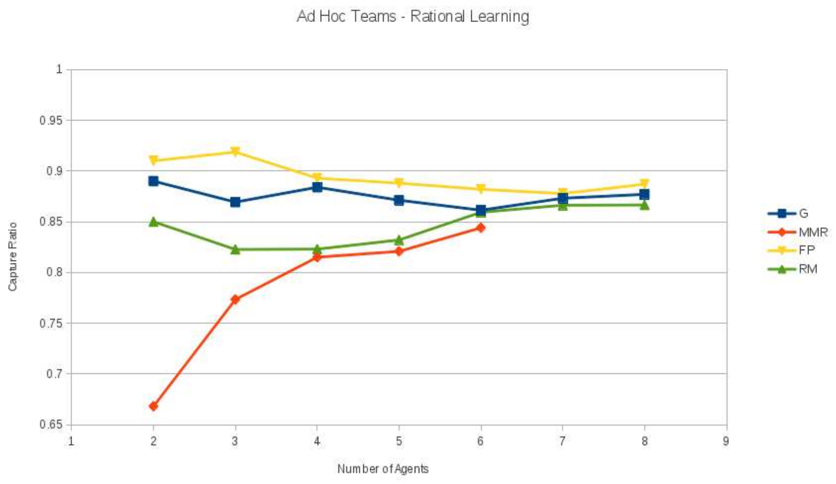


Figure 4.17: A visual representation of the individual results for ad hoc teams of rational learning predators from table 4.9. Each data point on each line summarizes 250 trials of that particular configuration.

4.8 Discussion

Throughout our series of experiments, we observed a wide array of interesting system behaviors and outcomes. In this section, we break apart these observations and piece together a broader picture of the algorithm as it relates to emergent cooperation in Defender and the field of multi-agent systems. We begin with a discussion of the emergent commitment behavior, comparing each learning algorithm's theoretical and empirical performance. This is followed by our comments on the various modifications to Defender and the optimizations to fictitious play and rational learning. With these ideas in place, we then move the discussion to the main difficulties that the problem presents and how our algorithm overcame them. Finally, we close the section with our thoughts on future work, namely the practical (robotic) implementations of these algorithms.

4.8.1 Emergent Commitment Behavior

We begin with the overall results that are summarized in tables 4.6 and 4.7. It is obvious that fictitious play and rational learning perform much better than the other algorithms. As a general solution to the Defender problem, action-based fictitious play yielded the best results, converging to a steady 85-90% approximate capture rate for predators. Figure 4.11 exemplifies the emergent commitment built from a combination of action-based fictitious play, a partially observable anti-coordination game, and an estimated distance-based payoff function.

Rational learning and fictitious play are related in their inherent attempt to build a probabilistic model of other predators' actions. They only differ in their method of updating the probabilistic model. Fictitious play takes the history of observations directly. Rational learning uses Bayesian updating to recompute the probability distribution of other predator's actions. The visual output looks very similar to fictitious play on the surface (see figure 4.11). We have observed that fictitious play provided a much smoother transition between mass points over other predators' actions in comparison to rational learning. This is because Bayesian updating tended to build a momentum in its updates, which caused a bit more impulsive decision shift in the model of other predators. Unfortunately, this led rational learning predators to more abruptly shift among possible prey with fewer observations of their teammates. Table 4.6 and figures 4.6 and 4.7 demonstrate the outcome of this impulsiveness: predictably lower performance. Nevertheless, both fictitious play and rational learning performed quite well in spite of receiving a very incomplete picture at each update step.

Interestingly, regret matching performed quite poorly in spite of its popularity among multi-agent researchers. The reason for this is quite simple: it uses utility to update its beliefs about available actions. We would like to preface our statements on regret matching with the understanding that these methods were not designed for utility-varying games, such as Defender. In fact, most game theoretic multi-agent learning algorithms are seldom investigated in dynamic scenarios. In any case, figure 4.12 clearly summarizes our observations on regret matching's behavior. By the third and fourth time step, it is apparent that regret matching predators are "confused" as to which prey they should chase. Let us break down equation 4.6 to grasp its shortcomings through a simple thought experiment. When the game begins, predators are far from their prey. This provides rewards that are relatively small in comparison to the maximum reward obtainable. As the game progresses, agents become much closer to one another. This causes all the rewards to rapidly increase. By the time agents are near one another, any previous decisions' rewards are very small in comparison to the large rewards for being in close proximity. In effect, the initial decisions apply almost no weight to the decisions once the predators are about to capture a prey. This in turn creates indecision, as any new prey that comes into view has a much higher reward associated with it. The reward-centered decision methods will all fall prey to this issue in dynamic, time-varying payoff games. We implemented this to clearly demonstrate the problem with these methods, e.g. regret matching, and hopefully spark new research to improve upon this multi-agent learning problem.

Our baseline methods also did not fare well in the dynamic scenario we presented. Both actually underscored the two extremes of pure commitment and pure indecision. The predators that used a greedy strategy were quite proficient in finding the best, i.e. closest, possible prey. Their downfall was that they committed to it indefinitely. Figure 4.13 captured this behavior in time steps four and five. The agents tended to all chase the same prey, with only one predator succeeding and the remaining left to pickup whatever prey were still in view. Minimax regret suffered from the similar kind of indecision found in regret matching. The difference was that the regret differed at every update so there was no consistency in the predator's decisions. Tables 4.6 and 4.7 show the outcome of this overabundance of commitment or indecision. There was one interesting point about the greedy strategy and its equivalence to fictitious play and rational learning. In a somewhat idealized scenario, fictitious play or rational learning predators may start acting at staggered times. If this occurs, the predators will only ever see the prey and perhaps one or two predators in front of them. This situation, as a result of the anti-coordination game, effectively creates a greedy strategy for the prey not being already chased.

The desired emergent commitment behavior for this kind of dynamic multi-predator multi-prey problem comes from three main components. The first component we selected was an adaptive learning algorithm, in particular fictitious play which is based on constructing probability distributions over other player's actions. One very important aspect was our decision to use action-based learning algorithms, instead of randomly selecting the action at each stage based on the probability distribution. We initially tried both and quickly found that randomization over probability distributions does *not* create emergent rapid commitment. By choosing to implement action-based algorithms, we ensure a degree of commitment based on the highest expected utility for the agent. The second component is an anti-coordination game. Our original reason for selecting anti-coordination game was to only reward actions that encouraged the even distribution of predators to prey. In other words, predators should receive a positive payoff for prey that are not already being chased, and a zero payoff otherwise. From this idea, we can already see how fictitious play might compliment this game structure, since the probability distribution of other predators' actions reflects the probability they no other predator is selecting the same prey. This was derived with equation 4.10. The third component is a payoff function centered around estimated travel distance. This is one of the common choices for utility in predator-prey problems; however, researchers have also used the probability of a capture instead. The capture probability was considered, but we realized that it too is related to the expected distance. Combining these two components with a learning algorithm in a game that is constantly changing in dimension (from the tracker's track additions and removals) produces our final algorithm. We have demonstrated its success with fictitious play in the experimentation section. In this section, we have discussed why it provides the desired emergent commitment behavior, mainly a result of the smooth transition between prey and easy adaptability amidst dynamism. The next sections will expand our understanding of what difficulties it has overcome and its robustness to changes in the game.

4.8.2 Difficulties in Dynamic Games

There are two main factors that contributed to the erroneous decisions made by all algorithms: the lack of knowledge about other agents in the game and the estimation function from the tracker. The former obviously contributes to a lot of issues, since unobservable predators may be much better candidates to chase a visible prey. Additionally, if the ideal prey lies just outside the field-of-view of a predator, there is no way for the predator to even incorporate that in its decision-making process. Furthermore, "negotiations" among predators, i.e. multiple observations of one another over time, are not able to occur without full knowledge of the quantities and states of predators and prey. This is an unavoidable problem with uncertainty, but as our experimentation has demonstrated, fictitious play agents that use a dynamic anti-coordination game in Defender can produce very good results regardless. In chapter 5, we will discuss the effects that distributed communication can have on the predators in an attempt to remedy the problem of lacking information.

Perhaps not as obvious as the problem of uncertainty, the estimation function found in the tracker is crucial to the success of the algorithm. The estimation function's role is simply to predict the prey that the predator is currently chasing. Game theoretic-inspired multi-agent learning algorithms operate over the actions performed by other players in a game. If those actions are consistently unreliable, then the learning algorithm can quickly break down. Therefore, our estimation function must mitigate the inaccuracies in the observed data to best predict a predator's actions. The Extended-Kalman Filter (EKF) aids in removing some of the sensor noise over time, but this alone is not always enough to allow for accurate estimations. We also found that increasing the update rate alleviates the problem to some degree, which was explored in section 4.7.3 and will be further discussed in section 4.8.3.

4.8.3 Modifications to the Environment and Team Composition

While it was not our main focus, we did investigate the effects that time update rates had on the predators. Figure 4.14 showed us that the percentage of prey captured improves as the update rate increases in the case of $n = m = 4$. We observed this same trend for other team sizes as well. This is important information since it tells us our algorithm is dependent on the update rate, but is still able to maintain a reasonable successful capture percentage even if the update rate is around 1 second. Therefore, we can say with confidence that our algorithm supports a processing overhead of 1 second. Additionally, we are able to show that performance does not decrease with faster update rates, in fact it only seems to improve. Lastly, increasing or decreasing the update rate is related to decreasing or increasing the engagement speeds, respectively. While they may not be equivalent, one could expect similar behavior in a modification of either.

The behavior can be observed from two angles: the actual game difference between updates and the number of samples for probability distribution approximation. Since the internal game represents a single snapshot in time of the true physical system, increasing the update rate creates smaller time windows. This helps eliminate the often abrupt changes in observation between the internal games' stages, approximately producing a repeated game in the limit. Shifting to the second view, the update rate is effectively the sample rate of some unknown probability distribution over actions for each predator. The more samples obtained, the better the approximation of the true distribution. Furthermore, increases to the sample rate remove a lot of the problems with noise. In both views, there is a positive correlation between increasing the update rate for predators and improving system performance.

We also experimented with the concept of ad hoc teams [81] in an effort to demonstrate robustness to various team compositions and deviations from our assumptions. Technically we already presented Defender as an ad hoc team setting, but before the experiments in section 4.7.4 we did not test adaptability to teammate predators using unknown algorithms. Our experimentation in this section also tried to bifurcate the similarities between fictitious play and rational learning. Tables 4.8 and 4.9 clearly show that the introduction of a regret-based agent severely hinders the system performance. As the team size increases, the negative effect of the single perturbed agent is phased out quite rapidly; once the standard predators outnumber the perturbed predators four-to-one, it tends to produce reasonable results again. The overall results for fictitious play and rational learning teams, as summarized by figure 4.15, clearly underscore that fictitious play is more robust to unknown changes in team composition. This further reinforces the evidence that action-based fictitious play is a great balance of adaptability and commitment, two properties that are very important for cooperative multi-agent systems.

4.8.4 Practical Implementations

Defender was conceived with the idea that the realistic physical applications of multi-agent systems have a great deal of uncertainty embedded within them. It attempts to model the most common forms of uncertainty. We built our algorithm consisting of a tracker, learning algorithm, anti-coordination game, etc. to produce an emergent cooperative "role" distribution behavior. For this reason, practical implementations of our algorithm and Defender itself should be fairly straightforward.

One issue that we have mentioned about game theoretic algorithms is that they often are quite computationally expensive. Our optimization for fictitious play and rational learning in section 4.6 sought to improve performance by leveraging the properties of our anti-coordination game. Figure 4.5 demonstrates the theoretical improvement that algorithm 4.6 could obtain without extra overhead, i.e. just leveraging the zeros. Equations 4.13 and 4.14 state the true computational complexity of both parts of the algorithm. In any case, the algorithm should be computationally feasible for direct use in real world systems.

For future work, we would like to implement Defender in a robotic setting and compare our empirical findings here with those of physical incarnations. We would also like to expand the use of our algorithm to other problems, such as the RoboCup. We believe that any dynamic role assignment scenario would benefit from our approach, as it was created with physical implementations in mind.

4.9 Summary

In this chapter, we presented the Defender problem. It is a multi-predator multi-prey problem that incorporates ideas from the current state of the art and competitive sports. Defender was constructed as an expanded version of the traditional predator-prey model that would enable the development of a distributed cooperative algorithm that would operate in highly dynamic environment requiring rapid convergence for success.

We began with a definition of Defender, followed by a discussion of our game structure. We chose an anti-coordination game because it encouraged an even distribution of predators to prey.

This included a payoff function that was proportional to the estimated path distance for the predator to capture a prey. With the game structure in place, we presented a brief example to point out some of the unusual properties of the highly dynamic environment. In particular, the game may change its dimension upon each update step. This led us to discuss the tracker, which is the source for this change in dimension. Our tracker maintains the set of tracks and observations that the predators use in their decision-making process. Once we understood the tracker's behavior and how it related to the anti-coordination game structure, we moved into a discussion of the learning algorithms.

For this thesis, we considered three popular algorithms founded in game theory: fictitious play, rational learning, and regret matching, in addition to two baseline algorithms: minimax regret and a greedy strategy for comparison. With the prescience from our experimentation section, we discussed the computational complexity of the algorithms and determined the number of zero multiplications in fictitious play and rational learning. The zero multiplications are effectively wasted computation time, so we built an optimized algorithm around this fact and showed theoretically and empirically that it outperforms the general algorithm many times over.

With the full understanding of our approach, we implemented Defender in a very detailed simulation. Our experimentation investigated all five learning algorithms and team sizes ranging from 1 to 8 agents. We also experimented with the update rate's effect on performance and the very relevant topic of ad hoc teams. Our discussion concluded with a collection of evidence that supported action-based fictitious play as the superior learning algorithm in these kinds of problems. We found anti-coordination games with payoff functions proportional to expected travel distance work quite well for distributing predators to prey in these constantly changing scenarios. Future multi-agent systems must be able to handle uncertainty and still meet the global objectives, a task for which we have shown our algorithm excels.

Commitment is one of the most important aspects of cooperating multi-agent systems [40]. Our algorithm applied to the Defender problem has shown that this behavior can arise naturally in these kinds of reinforced-payoff, dynamic, low information scenarios. We have taken a step towards bridging the often separated focuses of game theory and realistic applications containing uncertainty. This chapter has addressed the "moving the goalposts" problem in multi-agent systems [64]. We have demonstrated that relatively simple game theoretic algorithms can be quite robust to both adapt and commit so that the system converges to the specified global objectives.

Chapter 5

Ad Hoc Distributed Communication Architecture

In multi-agent systems, communication often plays a central role in the construction of agents. In our chapter on the Defender problem, we focused on solutions that used observations and inference. This chapter turns towards how *direct* communication may be created in dynamic systems. We will start with a description of our problem proceeded by a more formal problem statement. Next, we present the full distributed communication architecture and touch on some of its key elements. With problems like Defender in mind, we then discuss some optimizations for use in predator-prey-like domains. With the base algorithm and its extensions in place, we explain how it would apply to Defender. Then we present our results from experimentation, and conclude with a discussion of its overall behavior.

5.1 Description

We investigate the problem of creating a robust, rapidly converging communication architecture for low bandwidth, single channel problems such as RoboCup and Defender. To keep the algorithm as general as possible, we require only three minimal capabilities on the part of the agents. Our problem domain, predator-prey, benefits from having all agents equally share their knowledge since the lack of local information available to each agent is quite small. The more agents share their diverse world-views, the closer we can approximate a stable game. Therefore, we assume agents should communicate in a round-robin manner. Lastly, these problems do not have centralized controllers. Therefore, the algorithm must be completely distributed among the agents.

5.2 Problem Statement

Our distributed communication architecture works for single channel, low bandwidth scenarios. In this section we explicitly state the requirements on the parts of agents and the specific problem we will solve. We will also include a basic example to explain how the architecture leverages both high-level and low-level information to rapidly converge to a round-robin communication structure.

To remain as general as possible we enforce only minimal requirements. The problem domain is as follows:

Definition 23. *The problem domain of the* **Distributed Communication Architecture (DCA)** *is defined by the following properties:*

1. *A single channel, low bandwidth communication environment.*

2. *Agents require round-robin communication.*

3. *The number of agents trying to communicate is not known* a priori*.*

4. *Agents may drop in and out of communicating without prior notification.*

We would like to mention that these requirements are quite difficult to meet for the current state-of-the-art algorithms in multi-agent communication. If we relax any of the problem domain's requirements, it simplifies the problem. Thus, our algorithm would work with fewer restrictions, but there may be possibilities for further optimizations or other algorithms. With this particular problem domain, we now state our assumptions about each agent's capabilities in the system.

Definition 24. *The* **assumed agent's capabilities** *are:*

1. *Agents can classify a message to be from the set* {noise, silence, clarity}*.*

2. *Agents can determine if their own message was successfully transferred (*clarity*) or collided with another agent's message (*noise*).*

3. *Agents are able to understand one another.*

We have found that these assumptions cover almost all cooperative multi-agent systems. We also have included a variant which relaxes the third assumption. This variant does not allow for any of the messages to be understood by the agents in the system. Thus it only uses the pattern of {*noise, silence, clarity*} to build the distribution architecture. An example of this kind of scenario might be a domain in which agents must communicate using different encryption schemes. While this **non-sharing** variant will be covered briefly, we will mainly focus on the **sharing** version of our algorithm which requires at least one part of any message that is sent to be understood by all agents.

5.3 The Distributed Communication Architecture (DCA)

The general idea is to create a communication system for the unknown group of agents to each settle on a distinct time slot. The quantity of time slots will be designed to grow proportionally to the number of message collisions observed. This enables a mechanism to control the growth of time slots while simultaneously bounding the maximum number of times slots as a function of the number of agents. Before we delve into the details, let us first define a few terms we'll be using throughout the chapter.

Definition 25. *An agent's* **state** *is the current function it is running in the communication architecture, i.e. functions from the set* {learning, slot selection, settled}.

Definition 26. *A* **slot** *refers to the time-delimited window on the single channel (frequency) that an agent may speak. On a slot, agents recognize the type of signal as either* noise*,* silence*, or* clarity*.*

Definition 27. **Varying slots** *are slots that do not have an agent assigned to it. Agents attempt to send their messages on these slots.*

Definition 28. **Settled slots** *are slots that have an agent assigned to it. A settled agent will always speak on its respective settled slot.*

Definition 29. *A* **cycle** *is the concatenation of settled slots followed by varying slots. Therefore, the full cycle length varies by the size of each. See figure 5.1 for an example.*

As stated above, the algorithm divides itself into three states (see figure 5.1). Agents begin in the *learning* state, which only focuses on listening to other agents to count the number of varying and settled slots. Once the cycle is determined, the agent moves to the *slot selection* state. In this state, the agent repeatedly attempts to speak on random slots without any other agent speaking over it. Once a free slot is found, the agent moves to the *settled* state, which allows the agent to cyclically speak on that time slot.

5.3.1 Base Functions

We will define two interface functions for listening and speaking. The *listen function* is assumed to receive the pre-parsed input from the sensors in the form of either a specific message, noise, or silence. We also assume that the *speak function* outputs the desired message, whatever that may be for the agent.

In our algorithms, let the *state* variable refer to the current state of an agent as shown by figure 5.1. Agents maintain three variables: *currentSlot*, *mySlot*, and *numSlots*. These denote the current slot that the system is in, the current slot ID of the agent, and the total number of slots, respectively. Furthermore, each agent maintains a *slots* zero-indexed list, which describes both what was previously heard and the actual communication structure. Algorithm 5.1 describes the listen function and algorithm 5.2 describes the speak function.

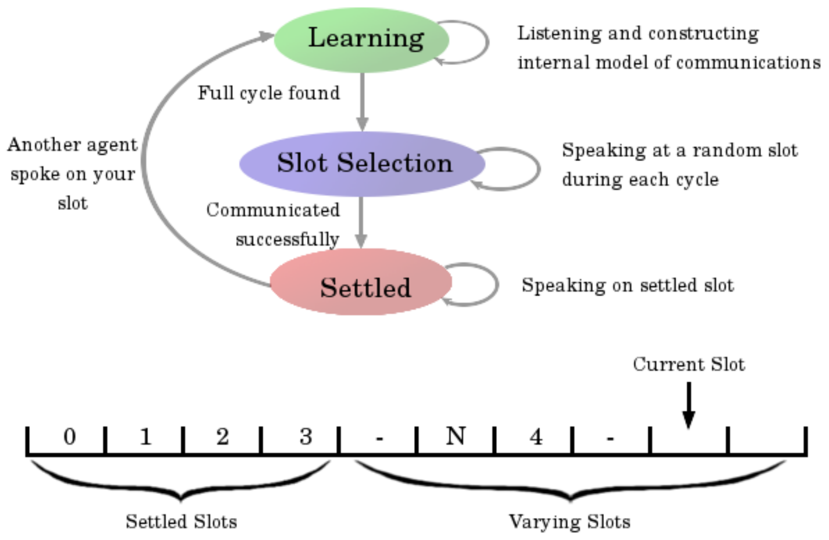

Learning
Listening and constructing internal model of communications
Full cycle found
Slot Selection
Speaking at a random slot during each cycle
Another agent spoke on your slot
Communicated successfully
Settled
Speaking on settled slot
Current Slot
0
1
2
3
-
N
4
-
Settled Slots
Varying Slots

Figure 5.1: The high-level state transition diagram for the communication algorithm (top). A visual example of the communication cycle structure (bottom) with numeric values showing the settled slot ID's, "N" representing a noisy signal, and "-" denoting a silent signal.

Algorithm 5.1 *listen*: Given a signal, update the internal state of an agent's communications architecture.

```
Require: $signal$: The new signal in $\{noise, silence, clarity\}$.
1: if $state$ = "$learning$" then
2:     $learning(signal)$
3: else if $state$ = "$slotselection$" then
4:     $slotSelection(signal)$
5: else if $state$ = "$settled$" then
6:     $settled(signal)$
7: end if
```

Recall that we have assumed a **sharing** system, in which agents must at least communicate their internal *mySlot* variable in the message. The **non-sharing** variant accounts for systems without this capability, but at the cost of slower convergence rates and more overhead. Furthermore, the sharing system allows for an automatically generated indexing system that agents can work out completely independently from a centralized numbering scheme. This may prove to be a tremendously valuable tool for self-referencing in messages and agent observation management.

5.3.2 Learning, Slot Selection, and Settled States

Now that the base structure is defined, we will provide details on the learning, slot selection, and settled states.

Algorithm 5.2 *speak*: Based on the current internal state, determine whether this agent should speak.

```
1: if $currentSlot = mySlot$ and $state \neq$ "$learning$" then
2:     return true
3: else
4:     return false
5: end if
```

5.3.2.1 The Learning State

The learning state's overall role is to determine the complete cycle, consisting of settled and varying slots, so that the agent may transition to the slot selection state as quickly as possible. The details for the **learning** state function are given in algorithm 5.3. This state has two distinct components: checking for pre-existing communication architecture and learning the established pattern.

Algorithm 5.3 *learning*: Learn the communication architecture that other agents are using.

```
Require: $signal$: The new signal in $\{noise, silence, clarity\}$.
1: Record the $signal$ in the next $slots$ list
2: if $signal$ = "$clarity$" then
3:     if the first slot has been heard twice with no noise in between then
4:         Count the number of slots, storing the result in $numSlots$
5:         Find the starting index and reorganize the $slots$ list
6:         Count the number of settled slots, storing the result in $numSettled$
7:         Count the number of varying slots, storing the result in $numVarying$
8:         Find and set the current slot variable, $currentSlot$
9:         Set the $mySlot$ variable equal to the end of the $slots$ list
10:        $state \leftarrow$ "$slotselection''$
11:        if $currentSlot = numSlots$ then
12:            Randomly select a new slot following equation 5.1
13:        end if
14:        $updateSlots(signal)$
15:        return
16:    else if the $signal$ is "clear" and the first slot ID, or this $signal$ was "noise" then
17:        Reset the $slots$ list to begin learning again
18:        $learn(signal)$
19:    end if
20: else if $signal$ = "$noise''$ then
21:    Reset the $slots$ list to begin learning again
22: end if
23: if it has been $maxWaitTime$ time steps since creation with only $signal$ = "$silence$" then
24:    Reset the $slots$ list
25:    $mySlot \leftarrow 0$
26:    $state \leftarrow$ "$slotselection$"
27: end if
```

The first component determines if a new communication architecture must be started or not. This is shown in the very last portion of the algorithm's psuedocode. The algorithm effectively waits a predefined number of time slots given by the constant *maxWaitTime*. If this time expires without any other agent being heard, the agent moves to the slot selection state with a *states* list consisting of one varying state. On the other hand, if the agent hears another agent, the second component is used. In this second component, the agent listens until it has heard the same agent speak twice. This will determine the full cycle, which may then be adjusted to find the first and last slot. It now ends up waiting until there is no noise in the cycle. It waits in this way to prevent the undesirable scenario in which a deluge of agents continue to flood the varying slots without agents ever settling. Eventually, it will be able to change the internal *state* to "slot selection," with the default attempt on the varying slots given by equation 5.1 from the next section.

5.3.2.2 The Slot Selection State

Since the full cycle has been determined, the agent begins to randomly attempt to find a settled slot amidst the varying slots available. This behavior is described by algorithm 5.4. In the random slot selection function below (equation 5.1), we refer to *numSettled* as the size of the set of settled states and *numVarying* to be the size of the set of varying states.

$$mySlot = numSettled + \lfloor \mathcal{U}(0, numVarying) \rfloor \tag{5.1}$$

In this slot selection state, only one attempt to find a free slot may be made in this way per cycle. Each agent's goal is to communicate its message without any other agent communicating its own at the same time, causing a collision. Upon success, the agent changes its *state* to be "settled" and updates the slots according to this change. If the agent fails, it will try again in the following cycle with a new randomly selection varying slot following equation 5.1. During this time, the *updateSlots* function manages the current list of settled and varying slots. This algorithm will be described in a later section.

Algorithm 5.4 *slotSelection*: Try to find a slot that no one else is using.

```
Require: $signal$: The new signal in $\{silence, noise, clarity\}$.
1: if $currentSlot = mySlot$ and $signal$ = "$clear$" then
2:     $state \leftarrow$ "$settled$"
3:     $updateSlots(signal)$
4:     return
5: end if
6: if $currentSlot = numSlots$ then
7:     Randomly select a new slot following equation 5.1
8: end if
9: $updateSlots(signal)$
```

5.3.2.3 The Settled State

Once an agent has settled on a state, it may communicate on that state at each cycle. It will also maintain the current cycle composition through the *updateSlots* function. This state also has a default mechanism to prevent the system from repeatedly causing message collisions. In the event of a collision, the agent in the settled state will move back to a learning state. The mechanism is provided to ensure any erroneous hardware issues don't cause system-wide problems and may be omitted, if desired. Algorithm 5.5 describes the settled state.

Algorithm 5.5 *settled*: This agent is settled so do not adjust anything unless another agent selects the same slot.

```
Require: $signal$: The new signal in $\{silence, noise, clarity\}$.
1: if $currentSlot = mySlot$ and $signal$ = "$noise$" then
2:    $state \leftarrow$ "$learning$"
3:    return
4: end if
5: $updateSlots(signal)$
```

5.3.3 Slot Organization

The *updateSlots* function was used in both algorithm 5.4 and 5.5. It is a crucial component of these algorithms, since it controls the addition and subtraction of settled and varying slots. The function is called every time a new signal is received to update the internal *slots* list for each agent. This procedure is given in algorithm 5.6.

Algorithm 5.6 *updateSlots*: Update a slot given a signal message and increment the current slot number.

```
Require: $signal$: The new signal which has just been heard in $\{silence, noise, clarity\}$.
 1: Store the $signal$ in the current slot
 2: Increment the $currentSlot$ variable
 3: if $currentSlot = numSlots$ then
 4:    Shift all previously settled slots down that were $noise$ or $silence$ this cycle, updating
       $numSettled$
 5:    Shift all previously varying slots that were $clarity$ this cycle, updating $numVarying$
 6:    Adjust $mySlot$ appropriately based on these shifts
 7:    Count the number of slots, storing the result in $numSlots$
 8:    Count the number of settled slots, storing the result in $numSettled$
 9:    Count the number of varying slots, storing the result in $numVarying$
10:    Count the number of silences ($numSilences$) and noises ($numNoises$) in the cycle
11:    if $numSilences = 0$ then
12:       Add a new silent slot, increasing both $numSlots$ and $numVarying$
13:    else
14:       $currentSlot \leftarrow 0$ (start a new cycle)
15:    end if
16: end if
```

During each stage in a cycle, this adds the signals to a list and maintains which is the current slot. Once the cycle comes to an end, it realigns the internal slots list by removing the agents who once were settled but did not communicate during their prescribed time slot. It also reorganizes all the newly settled agents from their varying slot to their new settled slot, ordered by their place in the cycle. Finally, it re-evaluates the internal state by checking if each varying slot was a collision, i.e. a slot with "noise;" then the number of varying slots will be expanded by one in the next cycle.

There is an underlying reason to resize the cycle only if there were no silences or equivalently only noise. Since we have enforced that the agents may only attempt to communicate once per cycle, we ensure that a full set of varying slots with noise means there exist at least double that number of agents in a slot selection state. This also prevents the number of slots in a cycle from growing infinitely due to repeated collisions. At a maximum, any given communication architecture will only have a number of agents equal to double the number of varying slots plus one, plus the number of settled slots. Other agents may be waiting in the learning state for the agents in a slot selection state to move to settled states.

5.4 Extensions

This algorithm is quite powerful as it has been presented so far; however, we may still improve upon it in two ways. First, we may include local agent knowledge in the algorithm to place more structure around the slot selection component, resulting in faster convergence rates. Second, we may examine the requirements for the non-sharing variant, removing the requirement for an agent's capability to parse a signal received.

5.4.1 Inclusion of Local Agent Knowledge

To improve the distributed communication algorithm's performance, we focus on the practical implementation of communication in predator-prey problems, such as Defender. In this variant, we embed an agent's local knowledge about the environment into the slot selection probabilities. This small adjustment has the potential to greatly improve performance, demonstrating an example of how to leverage local agent knowledge.

For the sake of simplicity, we design our conceptual model such that all agents are facing the same direction, they are traveling together in a 2-dimensional grid-world, and observations are made only of other agents in front of or next to them. This setup will only be used in our convergence rate experimentation. We apply the same algorithm to the highly dynamic Defender problem as well, with the same improved results. Figure 5.2 provides a visual example of this idealized model.

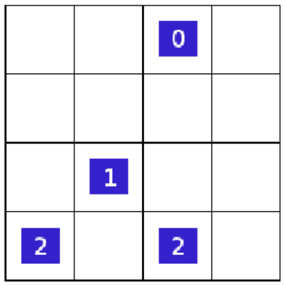


Figure 5.2: A visual example of the experiment for four agents, for which each agent can observe other agents strictly above it. The number of agents visible is shown as the number on each agent.

This scenario is very common in multi-agent systems as is evident in the RoboCup [47], predator-prey models [7], and pursuit-evasion problems [90]. For the equation below, let the number of agents visible for any agent be denoted by n.

$$mySlot = numSettled + \left\lfloor clamp(\mathcal{N}(\min(n, numVarying), \sqrt{numVarying}), 0, numVarying) \right\rfloor \tag{5.2}$$

In summary, this equation randomly selects the next slot attempt index by a normal distribution centered at the varying slot equal to the number of agents currently visible, with a standard deviation equal to the number of varying slots. This randomized result is clamped between the valid slots and converted to an integer (the slot index). The main idea is to select slots so that agents in front of the team have a predilection toward the lower varying indexes. Conversely, the agents which observe a lot of teammates will attempt to settle on varying slots with a higher index. In worst-case scenarios, agents who observe a lot of other agents will tend to collide with agents that also observe a lot of other agents, whereas the few agents that are in the front settle much more quickly. This creates emergent cooperative behavior in the system, and allows for a much more rapid creation of the communication pattern, as we will clearly demonstrate in section 5.6.

5.4.2 Non-Sharing Variant

As we mentioned before, the requirement that agents must be able to share their selected slot index can be relaxed. The non-sharing variant modifies the learning state (algorithm 5.3) to wait for two full cycles without any change in the number of settled slots, e.g. a structure like: $\{silence, clarity, clarity, slience, clarity, clarity, silence\}$. With a pattern such as this, an agent can guarantee the cycle size and that the system is prepared for new agents to join. This pattern now becomes the requirement to move from the learning state to the slot selection state.

There is one issue with this modification: this pattern can arise naturally if the last half of the slots are varying slots. While this event is quite rare, especially with large quantities of agents, it can occur. So our second modification enforces that all of the agents in the system speak on the final slot in the second cycle if this scenario arises. All the agents that are in a settled state or slot selection state know the true cycle; thus, they simply check for this pattern that the other learning agents might observe. The modification requires a simple two-cycle-long history check in the *speak* function (algorithm 5.2).

5.5 Application to Defender

We now examine the application of our distributed communication algorithm to the Defender problem with the hopes of improving performance. To do this, the message must be specified. In multi-agent systems, the messages are commonly hand-crafted for the particular problem; our approach to this is no different. We wanted to keep this in a moderately low bandwidth scenario. From our literature review, we found that as the bandwidth for communication increases, the problem begins to trivialize into a single-agent system. Additionally, these messages will only be shared once the predator has entered the settled state, so the possibility for an initial delay had to be considered as well.

We initially explored three message type ideas: observed predators, observed prey, and just the currently selected prey that the predator is chasing. Our rationale for sharing all observed predators is that the number of players in the internal game would be fully known, eliminating the key problem of losing the histories of selected actions. With the history intact, the learning algorithms would have a fair chance of utilizing all information to find the "best" cooperative solution. Similarly, the rationale for sharing all observed prey eliminates the key problem of missing the optimal prey to chase because it was only slightly outside the field-of-view. All prey would be considered, and the agent would then just adapt around the observable predators' actions.

The first two message types both required a great deal of state information about the full list of predators or prey, which goes against the idea of limited information. Furthermore, in the initial implementation of each we found overall performance to be lacking. Upon further investigation, we found a very interesting behavior. Before the predators created their distributed communication architecture, they approximated the algorithms found in chapter 4. This worked correctly, but once the architecture was formed and the predators began to send their observations to other agents, the system's "equilibrium" changed dramatically. This is due to the rapid increase and endurance in dimension of the internal games for either the set of predators or the set of prey. We concluded that the Defender problem did not provide enough time to readjust to the new game once the distributed communication architecture was put in place.

The suboptimal behavior of our first two attempts led us to examine our third message type: agents that share their state, including the current prey being chased. The idea came from our evaluation of the two primary causes of suboptimal behavior: the estimation function and the lack of knowledge. We had already attempted the ladder in two ways, so we addressed the former. Sharing the chased prey (i.e. action) enjoys a simple low bandwidth implementation, because the previous two required complicated management of tracks to include additional sporadic "observations" (in the form of messages). The communication addition takes the message from the agent who "spoke" and replaces any estimations made by the estimation function with the true location of the prey shared. If a track was already in place for that prey, it uses that track; otherwise, it rejects the message since they are not currently observable. The results we present in section 5.6 use this message type.

5.6 Experimentation

We provide a series of experiments to map the *convergence rates* of the general algorithm and modified algorithm that includes local knowledge. For each of these experiments, we considered the worst-case scenario in which all of the agents start simultaneously. This is the worst-case scenario because our algorithm will wait for all the agents in the system to move to a settled state before attempting to select a slot. In other words, they will just passively listen until it is their turn to try and speak. This means, that the *simultaneous start* scenario is a situation in which all agents begin at once, wait for the initial max waiting time, and then all simultaneously move into the slot selection state.

For completeness, we also investigated a kind of staggered scenario, in which agents are "ordered" to enter the system at a *random offset* in time from the previous one in the list. The random offset follows a discrete uniform distribution of $\lfloor \mathcal{U}(0, 10) \rfloor$ slots. We structured the experiment in this manner without loss of generality; since the agents are homogeneous, any permutation of them is an equivalent representation of the problem. Both the simultaneous start and random offset scenarios must be evaluated since the simultaneous start scenario does not fully account for the effects of all three communication states (learning, slot selection, and settled).

Convergence rates are not the only metric we used to evaluate performance. Since this is a communication system, we also examined the percentage of time slots that consisted of a clear signal per cycle, i.e. *clarity*, during the time spent constructing the emergent pattern. This metric essentially takes the number of clear messages in a cycle divided by the total number of slots in the cycle and takes averages over these intervals before convergence occurs. The resulting ratio represents the percentage of slots containing *message clarity*.

With the convergence rates in mind, we also experimented with communication in Defender. In particular, we applied the two kinds of sharing variants to our predator agents as described in section 5.5. Our rationale for our design came from the common low bandwidth scenarios found in the multi-agent literature. The goal for this inclusion of direct communication was to simply improve the performance rates we stated in chapter 4's experimentation section.

5.6.1 Convergence Rates

Due to space limitations, table 5.1 summarizes our results on convergence rates for 10, 20, 30, 40, and 50 agents. The full results for 1 to 50 agents are given in figures 5.3 and 5.4.

Experiment	10 Agents	20 Agents	30 Agents	40 Agents	50 Agents
Simultaneous Start	97.6 ± 19.2	330.6 ± 45.2	695.3 ± 82.5	1190.2 ± 111.3	1809.5 ± 155.6
Random Offset	94.9 ± 18.7	328.6 ± 47.3	697.4 ± 80.8	1194.8 ± 117.3	1816.6 ± 157.3
Simultaneous Start (K)	78.9 ± 21.7	264.8 ± 54.3	552.2 ± 89.5	961.5 ± 132.6	1481.5 ± 189.8
Random Offset (K)	94.6 ± 20.0	284.6 ± 50.5	570.5 ± 85.8	965.1 ± 126.5	1458.6 ± 173.1

Table 5.1: Given each type of experiment and the number of agents, the convergence rates (in number of steps) are shown. The (K) signifies that local agent knowledge was included.

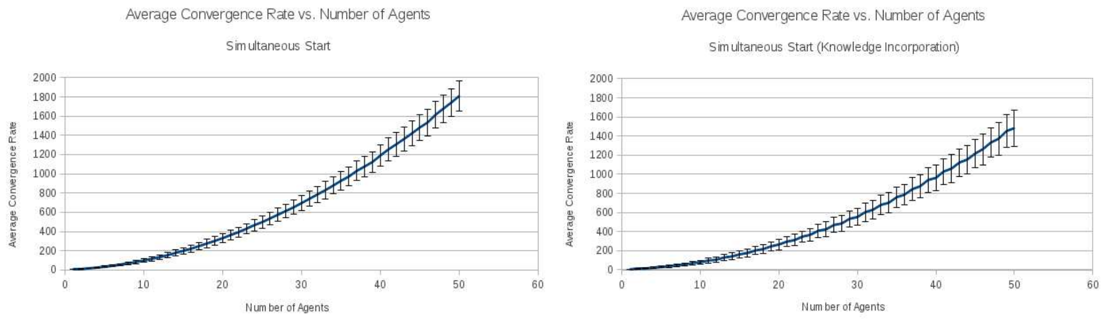


Figure 5.3: A visual of the convergence rates (in time slots) for 1 to 50 agents with a simultaneous start. The general algorithm's rates are given on the left figure, and the algorithm with agent knowledge is given on in right figure.

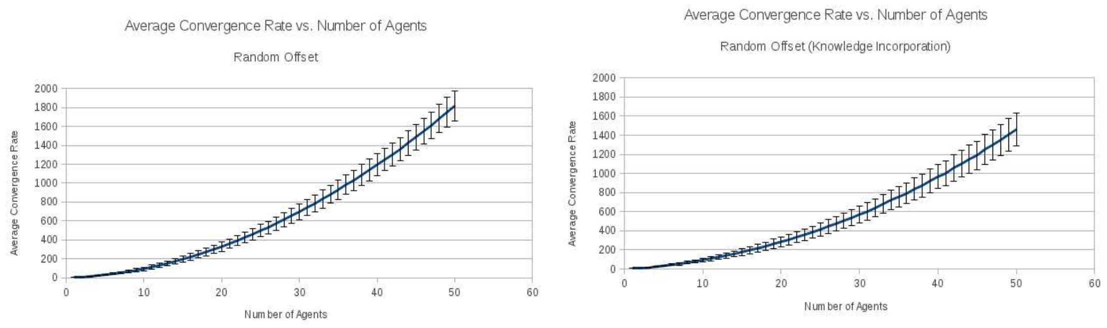


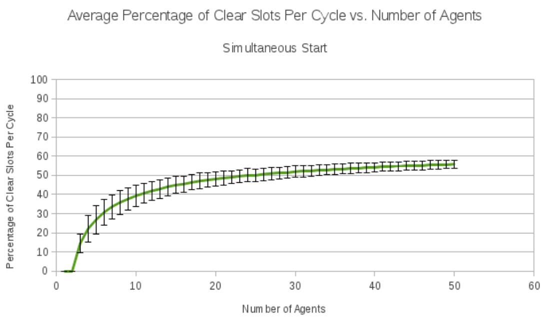
Figure 5.4: A visual of the convergence rates (in time slots) for 1 to 50 agents with a random offset. The general algorithm's rates are given on the left figure, and the algorithm with agent knowledge is given on in right figure.

5.6.2 Message Clarity Percentage

Table 5.2 summarize our results for the average proportion of message clarity per cycle before convergence occurs for 10, 20, 30, 40, and 50 agents. Figures 5.5 and 5.6 show 1 to 50 agents.

Experiment	10 Agents	20 Agents	30 Agents	40 Agents	50 Agents
Simultaneous Start	39.3% ± 5.5%	48.0% ± 3.7%	51.9% ± 3.0%	54.1% ± 2.5%	55.8% ± 2.2%
Random Offset	66.3% ± 5.4%	74.7% ± 3.5%	77.9% ± 2.7%	79.6% ± 2.2%	80.7% ± 2.2%
Simultaneous Start (K)	44.7% ± 7.9%	57.5% ± 5.0%	63.3% ± 3.6%	67.4% ± 2.8%	70.2% ± 2.5%
Random Offset (K)	66.8% ± 5.0%	74.1% ± 3.7%	78.2% ± 2.8%	80.3% ± 2.4%	82.1% ± 1.9%

Table 5.2: For each of the four types of experiments and number of agents, this table shows the average percentages of clear messages sent for a cycle. The (K) signifies that local agent knowledge was included.



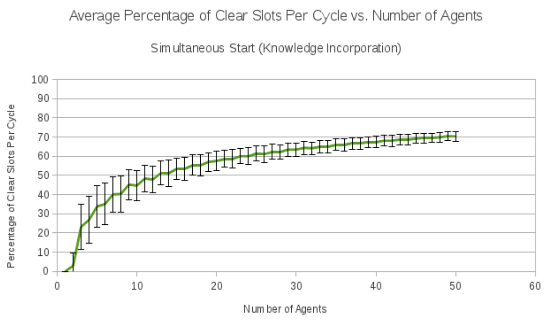
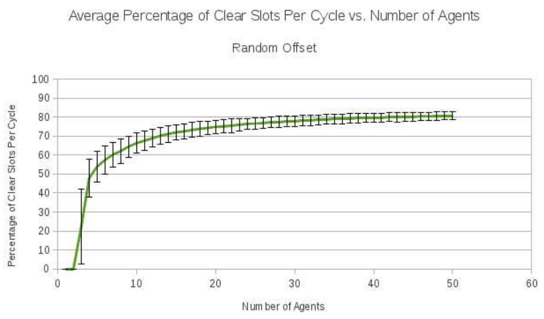


Figure 5.5: A visual of the proportion of clear messages (a percentage) for 1 to 50 agents with a simultaneous start. The general algorithm's proportions are given on the left figure, and the algorithm with agent knowledge is given on in right figure.



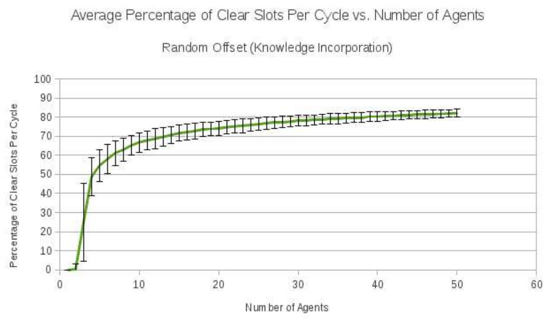


Figure 5.6: A visual of the proportion of clear messages (a percentage) for 1 to 50 agents with a random offset. The general algorithm's proportions are given on the left figure, and the algorithm with agent knowledge is given on in right figure.

5.6.3 Examples

To get a sense of how the algorithm behaves, we have included figures 5.7 and 5.8 that detail specific examples of the communication architecture formation.

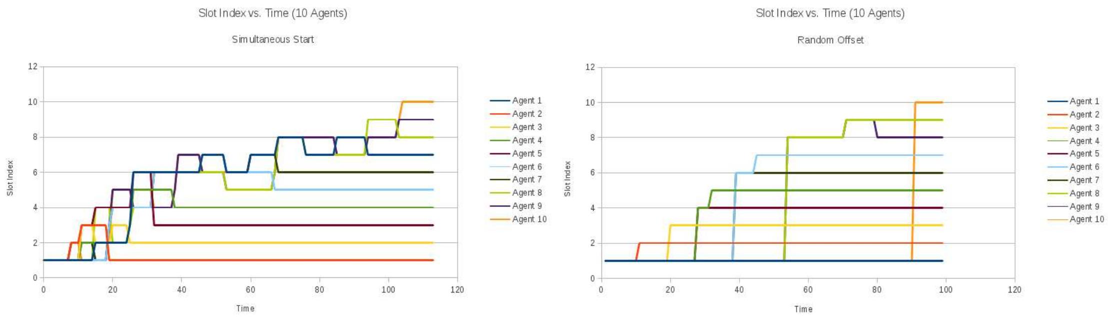


Figure 5.7: Two examples of the general algorithm in a worst-case scenario (left) and staggered scenario (right). Each example consists of 10 agents in the system. The axes denote the time step and slot, with each line representing the agent's slot choice history.

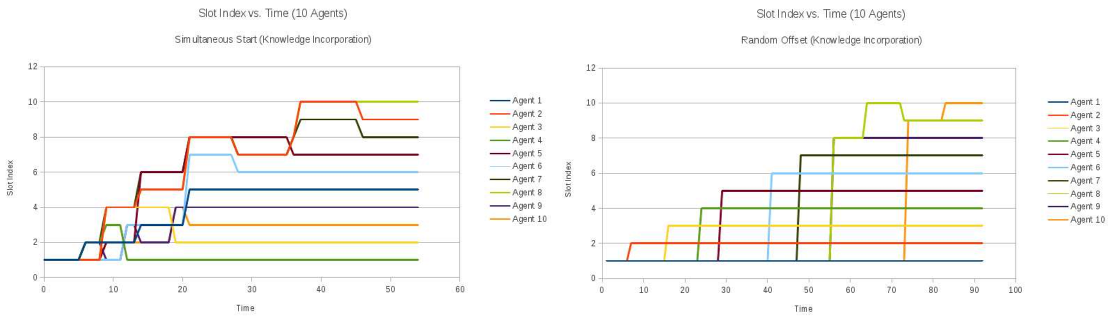


Figure 5.8: Two examples of the variant algorithm with knowledge included in a worst-case scenario (left) and staggered scenario (right). Each example consists of 10 agents in the system. The axes denote the time step and slot, with each line representing the agent's slot choice history.

5.6.4 Defender Communication Experiments

In a similar manner as our experimentation in chapter 4, we ran each permutation of the five predator types and two prey types with team sizes ranging from 1 to 6. Tables 5.3 and 5.4 summarize our results for the percentage of prey captured and the complete capture percentage, respectively. These are also described in figures 5.9, 5.10, 5.11, and 5.12.

	1	2	3	4	5	6
Simple						
Fictitious Play	98.8%	91.8%	88.0%	86.7%	86.1%	87.0%
Rational Learning	97.1%	93.2%	89.9%	88.8%	88.3%	87.4%
Regret Matching	99.1%	92.7%	88.6%	86.4%	82.9%	81.4%
Minimax Regret	99.4%	57.3%	49.4%	46.4%	n/a	n/a
Greedy	98.8%	88.9%	82.9%	81.0%	81.3%	79.9%
Maneuvering						
Fictitious Play	94.7%	90.6%	86.9%	85.7%	83.7%	83.2%
Rational Learning	96.5%	91.5%	89.1%	87.3%	85.9%	85.4%
Regret Matching	95.4%	89.9%	83.4%	82.0%	80.1%	78.5%
Minimax Regret	94.1%	65.9%	56.1%	53.5%	n/a	n/a
Greedy	91.6%	85.9%	81.8%	79.7%	80.3%	79.4%

Table 5.3: Results for the percentage of prey captured in configurations of {simple, maneuvering} prey vs. *communicating* {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 6 agents. These results summarize 1000 trials for each cell.

	1	2	3	4	5	6
Simple						
Fictitious Play	98.8%	84.1%	69.3%	56.8%	48.2%	43.3%
Rational Learning	97.1%	86.7%	71.0%	59.5%	50.5%	37.3%
Regret Matching	99.1%	85.5%	68.1%	54.1%	36.1%	25.2%
Minimax Regret	99.4%	35.2%	10.9%	04.6%	n/a	n/a
Greedy	98.8%	77.8%	50.9%	34.5%	27.3%	15.4%
Maneuvering						
Fictitious Play	94.7%	82.7%	66.4%	55.0%	43.2%	33.0%
Rational Learning	96.5%	83.3%	69.4%	55.4%	42.1%	33.0%
Regret Matching	95.4%	81.1%	56.8%	42.3%	29.1%	19.9%
Minimax Regret	94.1%	41.9%	16.7%	06.7%	n/a	n/a
Greedy	91.6%	71.8%	48.1%	32.6%	24.9%	16.4%

Table 5.4: Complete capture percentage results for configurations of {simple, maneuvering} prey vs. *communicating* {fictitious play, rational learning, regret matching, greedy, minimax regret} predators, given even teams of size 1 to 6 agents. These results summarize 1000 trials for each cell.

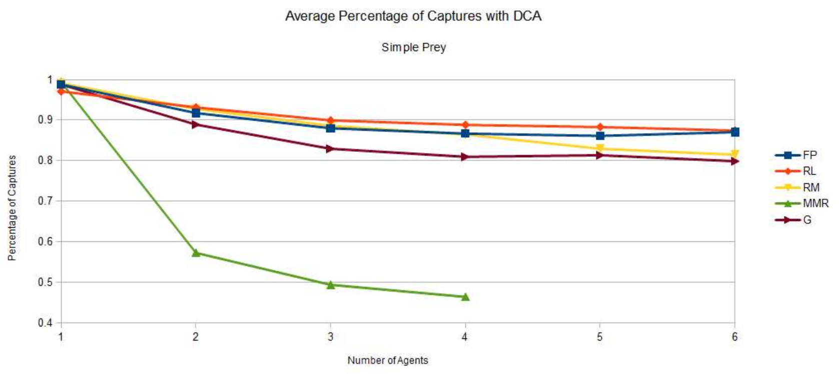


Figure 5.9: The percentage of simple prey captured. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6.

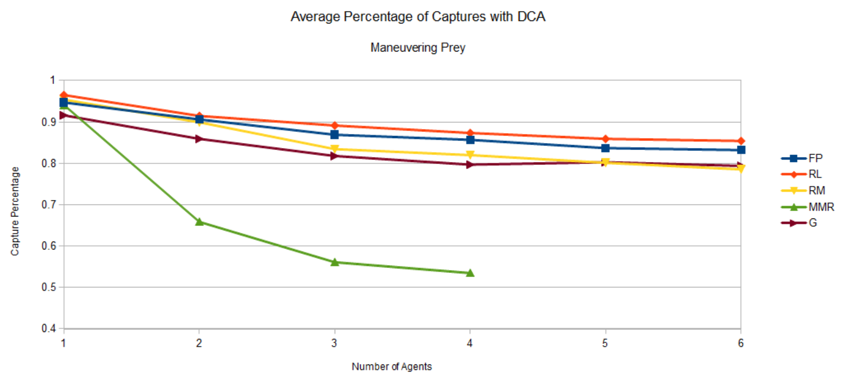


Figure 5.10: The percentage of maneuvering prey captured. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6.

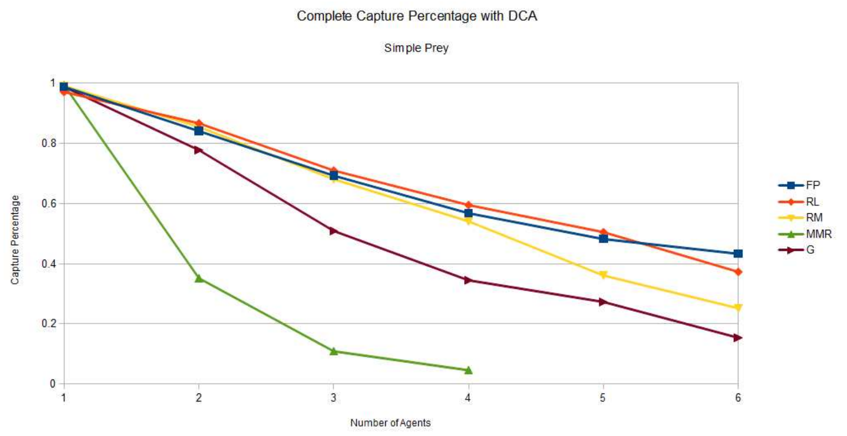


Figure 5.11: The complete capture percentage against simple prey. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6.

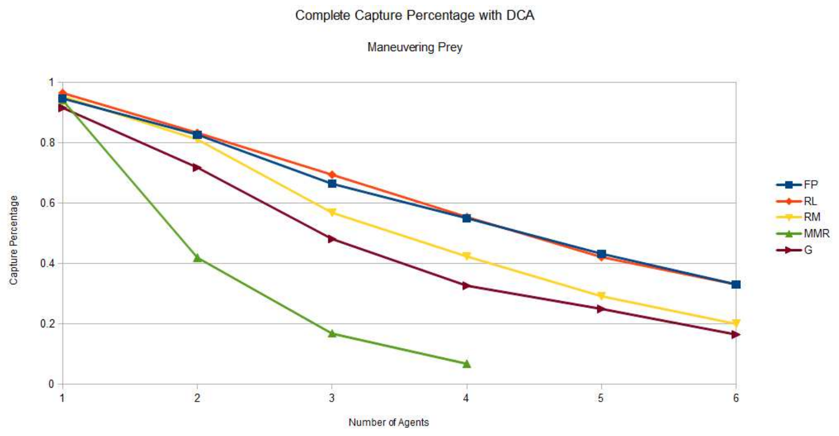


Figure 5.12: The complete capture percentage against maneuvering prey. This compares *communicating* fictitious play (FP), rational learning (RL), regret matching (RM), minimax regret (MMR), and greedy (G) agents with the number of agents ranging from 1 to 6.

5.7 Discussion

Upon first inspection of the results from our experiments, it is immediately apparent that there is an exponential growth in the convergence rates as a function of the number of agents. Table 5.1, and equivalently figures 5.4 and 5.3, both clearly show this behavior, hinting at the underlying structure of the algorithm. There is also a universally observed positive effect of including local agent knowledge. Convergence rates see upwards of a 25% increase in performance at 50 agents. The percentage of clear messages shows a different behavior: it seems to converge to a percentage. Table 5.2, and equivalently figures 5.6 and 5.5, state our results with this metric. They suggest a bound on performance with regard to the clear message proportion that is able to be sent before convergence. For the worst-case scenario of all agents starting simultaneously, percentage of clear slots per cycle remains at around 50% near 50 agents. If local agent knowledge is included, it reaches up to 70% at 50 agents. This pattern may in fact follow a logarithmic scale, which would imply approximately 100% as the number of agents $n \to \infty$; however, it is more likely to be converging since the algorithm depends on some *noise* and *silence*.

We may also compare the effect of staggering the start times with a random offset. The convergence rates were much faster in scenarios for which the agents began at different times. This is a useful result for the purposes of Defender and other realistic multi-agent systems (e.g. the RoboCup) since it is quite common for agents to want to begin communicating at different times. With the random offsets that were tested in section 5.6, we observed the same convergence rates for both simultaneous start and random offsets scenarios. Without knowledge, they had the same growth and reached 1800 total time steps on average without knowledge, and 1400 with knowledge. There was a performance improvement for the average percentage of clear messages. We saw a jump from 50% to 70% for simultaneous starts without and with knowledge included, respectively. Interestingly, we found the exact same performance for random offsets without and with knowledge included; both reached around 80% with 50 agents.

Another comparison may be made between the theoretical throughput of slotted ALOHA and our emperical throughput. We assume that the throughput of our system is equivalent to the average message clarity per cycle. This comparison is not always valid; our assumption examines the raw percentage of clear messages sent over an interval. In contrast, we might want to weight the importance of certain agents sending messages differently than others, which would not be equivalent to our average percentage clear messages per cycle. With this in mind, the theoretical throughput of slotted ALOHA is $1/e$, which is approximately 36.79%. Our method is not as efficient during the time it takes for our agents to converge with less that about 5 agents. After this point, we see that our approach has a higher percentage of clear messages sent. Therefore, our method outperforms a simple implementation of slotted ALOHA, with this particular metric's assumption. Moreover, if we examine our algorithm beyond the time until convergence occurs, then we will see it vastly outperform slotted ALOHA for all agent sizes since the agents will have 100% message clarity after the point of convergence. In summary, long-term communications will benefit greatly from DCA, whereas slotted ALHOA works better for smaller groups of agents that do not need to communicate more than a few messages.

The four examples we selected (figures 5.7 and 5.8) demonstrate the underlying reasons for our intriguing results. Comparing the difference between simultaneous start and random offset, we see that the random offset experiments have more clear messages being sent, i.e. less oscillatory behavior was observed between varying slots. Comparing knowledge inclusion or exclusion, it is evident that the two knowledge inclusion examples perform much better. This is a result of our decision to modify the slot selection state's randomization equation. The emergent behavior can actually be seen in the left example from figure 5.8. Some of the agents tend to randomize around lower slots and some attempt higher slots, specifically in the 10 to 20 time step region. These results clearly demonstrate the value of including local agent knowledge to improve performance.

With a better understanding of the convergence properties, we now turn to the trials on communicating agents in Defender. A previously stated, our initial results for other message types were always inferior to those of the non-communicating predators. Sharing the agent's location and the current prey it is chasing provided the best results. Tables 5.3 and 5.4 clearly show that fictitious play and rational learning still perform the best overall. Interestingly, it seems that with communications of this type, rational learning is slightly better than fictitious play. The is even more clear in figures 5.11, 5.9, 5.12, and 5.10.

These results were slightly worse with communication, in large part due to the steady increase in knowledge about the system. This increase causes the dimension of the internal matrix games to increase as well. So as the dimension increases, the time required to reach an "equilibrium" or "commit" increases beyond the time available to distribute the predators themselves to specific prey. It is these extra adjustments that doom the agents to the sub-desirable performance we observed.

5.8 Summary

This algorithm attempts to solve the problem of single channel, low bandwidth, distributed communication in multi-agent systems. The Distributed Communication Architecture (DCA) creates an emergent round-robin pattern of communication. It handles any number of agents, random start times, and allows agents to drop out at will; therefore, this algorithm is quite general. We demonstrated its success in a number of rigorous trials to test both its convergence rate and efficiency during construction (i.e. percentage of clear messages per cycle). Furthermore, we tested its performance in the Defender problem to provided an example of a practical application, in spite of its average net effect on performance in this particular scenario. We also presented two variants: one that relaxes the assumption that agents can share information, and one that includes local agent knowledge. We implemented the knowledge variant and also ran our experiments on it. The results showed as high as a 25% increase in convergence rates over the general algorithm, clearly confirming the applicability to real-world problems. In summary, we have presented a fully decentralized algorithm for multi-agent systems that creates a dynamic but structured round-robin communication architecture.

Chapter 6

Conclusion and Future Work

In this thesis, we have investigated various perspectives on solving the problem of multi-agent coordination in highly dynamic scenarios. We began with a survey of the current state of the art in chapter 2, noting that there is a growing trend towards the issues covered in this thesis. We then introduced a brief background of the relevant material in chapter 3. Next, chapter 4 investigated the Defender problem, a variant of the traditional predator-prey problem that requires rapid cooperation and commitment on behalf of the predators to capture each prey. We used a time-varying anti-coordination game with payoffs proportional to the estimated travel distance between agents. We also compared five kinds of well-known algorithms and optimized the performance of the two highest performing ones. Finally, Chapter 5 presented a distributed communication architecture for use in these cooperative information-sparse problems. It uses a logical approach to select time-slots so that agents move from a learning state to a slot selection state and then finally to a settled state.

In chapter 4's experiments on Defender, we found that fictitious play performed the best overall. In general, it averages an 85% to 90% capture percentage for our specific simulations. Each learning algorithm was evaluated over 5000 trials for each configuration of 1 to 8 agents and simple or weaving prey. Furthermore, we showed that it is much more viable in an ad hoc team setting and modeled the effects that update rate has on its performance. In these series of experiments, we found that regret-based methods do not work well for dynamic environments, whereas probabilistic agent modeling does, in spite of the constantly changing game. We found two main issues that the proposed algorithms overcome: the tracker's loss of tracks, and estimating which action each observed predator took. In congress with our experimentation, we saw a need to optimize fictitious play and rational learning so we also developed an optimized version and did a complete analysis on its computational complexity both theoretically and empirically. Overall, our simulations showed that emergent cooperation can arise in these highly dynamic, information-sparse, role assignment multi-agent scenarios.

Our Distributed Communication Architecture (DCA) from chapter 5 stated a robust algorithm for round-robin communication without the need for any centralized control. With our aforementioned slot-based approach, we evaluated the convergence rates from 1 to 50 agents, clearly demonstrating its viability in both worst-case and common-case situations. We also introduced a variant for predator-prey problems that enables agents to include their local state knowledge into the slot selection state. We evaluated each of the configurations' performance and found that for 10 agents the convergence rate is approximately 97 steps for the worst-case without knowledge but approximately 78 steps with knowledge included. The percentage of clear messages were 39 and 44 for without knowledge and with knowledge, respectively. This pattern was shown to be consistently true for all configurations. Next, we implemented communications in Defender but found that performance did not improve at all; in fact, it was slightly worse. Interestingly, our experimentation showed that communication increased the amount of "deliberation" among the agents before stability emerged. Therefore, we concluded that information-sparsity is actually beneficial for rapid adaptation, a property that matches exactly with the game theoretical concepts behind correlated equilibria.

Future work on these ideas include a robotic incarnation of the Defender problem to further validate our claims. We would also like to implement a few more game theoretic agents, e.g. auction-centered agents, to see if there are other solutions. We would like to construct a general mathematical model that describes Defender and formally prove theorems relating to the distribution of agents to roles in these dynamic environments. For our distributed communication algorithm we would like to find a closed-form solution for the convergence rate. We would also like to develop an indirect communication algorithm to provide an inference mechanism for agents in these problems. In summary, there are many directions to further develop our work which seeks to understand and address the issues of uncertainty that arise from the dynamism in realistic multi-agent systems.

Appendix A

Definitions and Theorems

A.1 Probability Distributions

We assume a basic knowledge of probability theory. For completeness, definitions 30, 31, 32, and 33 present the pertinant information for the uniform, categorical, Dirichlet, and normal distributions, respectively.

Definition 30. *The* **uniform distribution** $\mathcal{U}(a,b)$ *is defined with a probability mass function:*

$$f(x,a,b) = \begin{cases} \frac{1}{b-a} & \text{if } a < x < b \\ 0 & \text{otherwise} \end{cases}$$

It has a mean of $E[X] = \frac{1}{2}(a+b)$.

Definition 31. *The* **categorical distribution** $Cat(\vec{p})$ *is defined with a probability mass function:*

$$f(x,\vec{p}) = \prod_{i=1}^{k} p_i \cdot \mathcal{I}\{x = i\}$$

for each p_i *describing the probability of "success" for category* i. *It requires that* $\sum_{i=1}^{k} p_i = 1$ *and has a mean of* $E[X = i] = p_i$.

Definition 32. *The* **Dirichlet distribution** $Dir(\vec{\alpha})$ *is defined with a probability mass function:*

$$f(\vec{x},\vec{\alpha}) = \frac{1}{B(\vec{\alpha})} \prod_{i=1}^{k} x_i^{\alpha_i - 1}$$

with $B(\cdot)$ *as the Beta function. It requires* $\alpha_1, \ldots, \alpha_k, x_1, \ldots, x_k > 0$ *and* $\sum_{i=1}^{k} x_i < 1$. *Its mean is* $E[X = i] = \frac{\alpha_i}{\sum_{j=1}^{k} \alpha_j}$.

Definition 33. *The* **normal distribution** $\mathcal{N}(\mu, \sigma^2)$ *is defined with a probability mass function:*

$$f(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{\frac{-1}{2}(\frac{x-\mu}{\sigma})^2\}$$

with a mean of $E[X] = \mu$ *and variance of* $Var[X] = \sigma^2$.

Our application of Bayes' Rule in Rational Learning leverages the knowledge that the conjugate prior of a Dirichlet distribution is the categorical distribution. We state this in theorem 1.

Theorem 1. *The Dirichlet distribution is the conjugate prior for the categorical distribution.*

Appendix B

Predator Pseudocode

Within this appendix we present the five predators' pseudocode. These are discussed in chapter 4.

Algorithm B.1 *fictitiousPlayPredator*: The fictitious play predator.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $P$: A probability matrix, with $P_{ij}$ denoting the probability predator $i$ selects prey $j$.
Require: $\Omega$: A set of observations (action indices), with $\Omega_i$ denoting the observations of predator $i$.
Require: $a_{prev}$: The previously selected prey, if any.
Require: $E[a_{prev}]$: The expected utility for the previous prey at this stage, and 0 if no prey selected.
// Update the probability matrix with the new observation.
for $i \in \{0, \ldots, n-1\}$ do
    for $j \in \{0, \ldots, m-1\}$ do
        $P_{ij} \leftarrow 0$
        for $\omega \in \Omega_i$ do
            if $\omega == j$ then
                $P_{ij} \leftarrow P_{ij} + 1$
            end if
        end for
        $P_{ij} \leftarrow P_{ij}/|\Omega_i|$
    end for
end for
// Determine the next action.
$a^* \leftarrow a_{prev}$
$E[a^*] \leftarrow E[a_{prev}]$
for $a_i \in \{0, \ldots, m-1\}$ do
    $p^*(a_i) \leftarrow 0$
    for $a_{-i} \in \{0, \ldots, m^n - 1\}$ do
        $p^*(a_i, a_{-i}) \leftarrow 1$
        for $a' \in \{0, \ldots n-1\}$ do
            $\hat{a} \leftarrow resolveAction(n, m, a_{-i}, a')$
            if $\hat{a} = a_i$ then
                $p^*(a_i, a_{-i}) \leftarrow p^*(a_i, a_{-i}) * 0$
            else
                $p^*(a_i, a_{-i}) \leftarrow p^*(a_i, a_{-i}) * P_{a',\hat{a}}$
            end if
        end for
        $p^*(a_i) \leftarrow p^*(a_i) + p^*(a_i, a_{-i})$
    end for
    $E[a_i] \leftarrow u_i(a_i) * p^*(a_i)$
    if $E[a_i] > E[a^*]$ then
        $a^* \leftarrow a_i$
        $E[a^*] \leftarrow E[a_i]$
    end if
end for
$a_{prev} \leftarrow a^*$
$E[a_{prev}] \leftarrow E[a^*]$
return $a^*$
```

Algorithm B.2 *rationalLearningPredator*: The rational learning predator.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $P$: A probability matrix, with $P_{ij}$ denoting the probability predator $i$ selects prey $j$.
Require: $\Omega$: A set of observations (action indices), with $\Omega_i$ denoting the observations of predator $i$.
Require: $a_{prev}$: The previously selected prey, if any.
Require: $E[a_{prev}]$: The expected utility for the previous prey at this stage, and 0 if no prey selected.
 1: // Update the probability matrix with the new observation.
 2: for $i \in \{0, \ldots, n-1\}$ do
 3:   $d \leftarrow 0$
 4:   for $j \in \{0, \ldots, m-1\}$ do
 5:     $P_{ij} \leftarrow p(\Omega_{ij}|j)p(j)$
 6:     $d \leftarrow d + p(\Omega_{ij}|j)p(j)$
 7:   end for
 8:   $P_{ij} \leftarrow P_{ij}/d$
 9: end for
10: // Determine the next action.
11: $a^* \leftarrow a_{prev}$
12: $E[a^*] \leftarrow E[a_{prev}]$
13: for $a_i \in \{0, \ldots, m-1\}$ do
14:   $p^*(a_i) \leftarrow 0$
15:   for $a_{-i} \in \{0, \ldots, m^n - 1\}$ do
16:     $p^*(a_i, a_{-i}) \leftarrow 1$
17:     for $a' \in \{0, \ldots n-1\}$ do
18:       $\hat{a} \leftarrow resolveAction(n, m, a_{-i}, a')$
19:       if $\hat{a} = a_i$ then
20:         $p^*(a_i, a_{-i}) \leftarrow p^*(a_i, a_{-i}) * 0$
21:       else
22:         $p^*(a_i, a_{-i}) \leftarrow p^*(a_i, a_{-i}) * P_{a',\hat{a}}$
23:       end if
24:     end for
25:     $p^*(a_i) \leftarrow p^*(a_i) + p^*(a_i, a_{-i})$
26:   end for
27:   $E[a_i] \leftarrow u_i(a_i) * p^*(a_i)$
28:   if $E[a_i] > E[a^*]$ then
29:     $a^* \leftarrow a_i$
30:     $E[a^*] \leftarrow E[a_i]$
31:   end if
32: end for
33: $a_{prev} \leftarrow a^*$
34: $E[a_{prev}] \leftarrow E[a^*]$
35: return $a^*$
```

Algorithm B.3 *regretMatchingPredator*: The regret matching predator.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $\Omega$: A set of observations (action indices), with $\Omega_i$ denoting the observations of predator $i$.
Require: $a_{hist}$: The history of actions performed by this agent, with $a_{hist}^{(t)}$ representing the action performed at stage $t$.
1: $a^* \leftarrow a_{hist}^{(t-1)}$
2: $n(\cdot) \leftarrow 0$
3: $d \leftarrow 0$
4: for $a' \in \{0, \ldots, m-1\}$ do
5:   $r \leftarrow 0$
6:   $r(\cdot) \leftarrow 0$
7:   for $t \in \{0, \ldots, |\Omega_{a'}| - 1\}$ do
8:     $r \leftarrow r + u_i^{(t)}(a_{hist}^{(t)})$
9:     $r(a') \leftarrow r(a') + u_i^{(t)}(a')$
10:  end for
11:  $r \leftarrow r/|\Omega_{a'}|$
12:  $r(a') \leftarrow r(a')/|\Omega_{a'}|$
13:  $n(a') \leftarrow max\{r(a') - r, 0\}$
14:  $d \leftarrow d + n(a')$
15: end for
16: $random \leftarrow U(0,1)$
17: $step \leftarrow 0$
18: for $a' \in \{0, \ldots, m-1\}$ do
19:   $step \leftarrow step + n(a')/d$
20:   if $step \geq random$ then
21:     $a^* \leftarrow a'$
22:     break
23:   end if
24: end for
25: return $a^*$
```

Algorithm B.4 *minimaxRegretPredator*: The minimax regret predator.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $P$: A probability matrix, with $P_{ij}$ denoting the probability predator $i$ selects prey $j$.
Require: $a_{prev}$: The previously selected prey, if any.
Require: $E[a_{prev}]$: The expected utility for the previous prey at this stage, and 0 if no prey
   selected.
$a^* \leftarrow a_{prev}$
$E[a^*] \leftarrow E[a_{prev}]$
$z \leftarrow \infty$
for $a_i \in \{0, \ldots, m-1\}$ do
   $x \leftarrow 0$
   for $j \in \{0, \ldots, m^n - 1\}$ do
      $a_{-i} \leftarrow \emptyset$
      for $k \in \{0, \ldots, n-1\}$ do
         $a_{-i} \leftarrow a_{-i} \bigcup resolveAction(n, m, j, k)$
      end for
      $r \leftarrow 0$
      for $a'_i \in \{0, \ldots, m-1\}$ do
         if $r < u_i(a'_i, a_{-i})$ then
            $r \leftarrow u_i(a'_i, a_{-i})$
         end if
      end for
      $r \leftarrow r - u_i(a'_i, a_{-i})$
      if $x < r$ then
         $x \leftarrow r$
      end if
   end for
   if $x < z$ then
      $z \leftarrow x$
      $a^* \leftarrow a_i$
   end if
end for
return $a^*$
```

Algorithm B.5 *greedyPredator*: The greedy predator.

```
Require: $n$: The number of observed predators at this stage.
Require: $m$: The number of observed prey at this stage.
Require: $a_{prev}$: The previously selected prey, if any.
Require: $E[a_{prev}]$: The expected utility for the previous prey at this stage, and 0 if no prey
    selected.
 1: $a^* \leftarrow a_{prev}$
 2: $E[a^*] \leftarrow E[a_{prev}]$
 3: for $a_i \in \{0, \ldots, m-1\}$ do
 4:     $E[a_i] \leftarrow u_i(a_i)$
 5:     if $E[a^*] < E[a_i]$ then
 6:         $a^* \leftarrow a_i$
 7:         $E[a^*] \leftarrow E[a_i]$
 8:     end if
 9: end for
10: return $a^*$
```

Bibliography

[1] N. Abramson. The throughput of packet broadcasting channels. *IEEE Transactions on Communications*, 25(1):128, 1977.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[3] M. Alighanbari and J.P. How. Cooperative task assignment of unmanned aerial vehicles in adversarial environments. In *American Control Conference, 2005. Proceedings of the 2005*, volume 7, June 2005.

[4] G. Arslan, J.R. Marden, and J.S. Shamma. Autonomous vehicle-target assignment: A game theoretical formulation. *ASME Journal of Dynamic Systems, Measurement, and Control*, 129:584–596, September 2007.

[5] S. Barrett, P. Stone, and S. Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proceedings of Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, May 2011.

[6] A.G. Barto, R.S. Sutton, and C.J.C.H Watkins. Learning and sequential decision making. In *Learning and Computational Neuroscience*, pages 539–602. MIT Press, 1989.

[7] M. Benda, V. Jagannathan, and R. Dodhiawalla. On optimal cooperation of knowledge sources - an emperical investigation. Technical Report BCS-G2010-28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, July 1986.

[8] D.S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 32–37, 2002.

[9] P.B. Bhat, V.K. Prasanna, and C.S. Raghavendra. Adaptive communication algorithms for distributed heterogeneous systems. In *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 310–321, July 1998.

[10] C. Boutilier. Learning conventions in multiagent stochastic domains using likelihood estimates. *Uncertainty in Artificial Intelligence*, pages 106–114, 1996.

[11] M. Bowling. Convergence and no-regret in multiagent learning. In *In Advances in Neural Information Processing Systems 17*, pages 209–216. MIT Press, 2005.

[12] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *Autonomous Agents and Multi-Agent Systems*, 19:297–331, December 2009.

[13] G.W. Brown. Iterative solutions of games by fictitious play. *Activity Analysis of Production and Allocation*, pages 374–376, 1951.

[14] A. Campbell and A. Wu. Multi-agent role allocation: issues, approaches, and multiple perspectives. *Autonomous Agents and Multi-Agent Systems*, 22:317–355, 2011.

[15] J. Denzinger and M. Fuchs. Experiments in learning prototypical situations for variants of the pursuit game. In *International Conference on Multiagent Systems*, pages 48–55, 1995.

[16] E.H. Durfee and T.A. Montgomery. Mice: A flexible testbed for intelligent coordination experiments. In *Proceedings of the 1989 Distributed AI Workshop*, pages 25–40, 1989.

[17] T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *Proceedings of the Third International Conference on Information and Knowledge Management*, CIKM '94, pages 456–463. ACM, 1994.

[18] A. Friedman. *Differential games.* Pure and applied mathematics. Wiley-Interscience, 1971.

[19] D. Fudenberg and D.M. Kreps. Learning mixed equilibria. *Games and Economic Behavior*, 5(3):320–367, 1993.

[20] D. Fudenberg and D.K. Levine. Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, 19(5-7):1065–1089, 1995.

[21] D. Fudenberg and D.K. Levine. Conditional universal consistency. *Games and Economic Behavior*, 29(1-2):104–130, 1999.

[22] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, 1991.

[23] B.P. Gerkey and M.J. Mataric. Principled communication for dynamic multi-robot task allocation. *Experimental Robotics VII, LNCIS 271*, pages 353–362, 2001.

[24] B.P. Gerkey and M.J. Mataric. Sold!: auction methods for multirobot coordination. *Robotics and Automation, IEEE Transactions on*, 18(5):758–768, October 2002.

[25] B.P. Gerkey and M.J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[26] B.P. Gerkey and M.J. Mataric. On role allocation in robocup. In *RoboCup 2003: Robot Soccer World Cup VII*, pages 43–53. Springer, 2004.

[27] B.P. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. In *International Journal of Robotics Research*, pages 20–27, July 2004.

[28] P.J. Gmytrasiewicz and E.H. Dufree. Rational communication in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 4:233–272, 2001.

[29] P.J. Gmytrasiewicz and E.H. Durfee. Rational coordination in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 3:319–350, December 2000.

[30] C.V. Goldman and S. Zilberstein. Decentralized control of cooperative systems: Categorization and complexity analysis. *Journal of Artificial Intelligence Research*, 22:143–174, November 2004.

[31] L. Guibas, J. Latombe, S. Lavalle, D. Lin, and R. Motawni. Visibility-based pursuit evasion in a polygonal environment. In *Algorithms and Data Structures*, volume 1272 of *Lecture Notes in Computer Science*, pages 17–30. Springer Berlin / Heidelberg, 1997.

[32] E.L. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in Communications*, 9:1024–1039, 1991.

[33] I. Harmati and K. Skrzypczyk. Robot team coordination for target tracking using fuzzy logic controller in game theoretic framework. *Robotics and Autonomous Systems*, 57:75–86, January 2009.

[34] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[35] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. 19(4):281–316, 2005.

[36] J. Hu and M.P. Wellman. Online learning about other agents in a dynamic multiagent system. In *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, pages 239–246, New York, NY, USA, 1998. ACM.

[37] Y. Hu and V.O.K. Li. Satellite-based internet: a tutorial. *IEEE Communications Magazine*, 39(3):154–162, March 2001.

[38] A. Jafari, A.R. Greenwald, D. Gondek, and G. Ercal. On no-regret learning, fictitious play, and nash equilibrium. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 226–233, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[39] I. Jawhar and J. Wu. A race-free bandwidth reservation protocol for qos routing in mobile ad hoc networks. *Ad Hoc and Sensor Wireless Networks*, 1:1–28, 2005.

[40] N.R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *Knowledge Engineering Review*, 8:223–277, 1993.

[41] L.B. Jiang and S.C. Liew. An adaptive round robin scheduler for head-of-line-blocking problem in wireless lans. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 2, pages 1219–1224, March 2005.

[42] E. Kalai and E. Lehrer. Rational learning leads to nash equilibrium. *Econometrica*, 61(5): 1019–1045, 1993.

[43] S. Kalam and M. Gani. Non-cooperative target assignment using regret matching. In *Control Automation Robotics Vision (ICARCV), 2010. Proceedings of the Eleventh International Conference on*, pages 787 –792, December 2010.

[44] S. Kalam, M. Gani, and L. Seneviratne. A game-theoretic approach to non-cooperative target assignment. *Robotics and Autonomous Systems*, 58:955–962, August 2010.

[45] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, pages 35–45, March 1960.

[46] S. Keshmiri and S. Payandeh. Toward opportunistic collaboration in target pursuit problems. In *Second International Conference on Autonomous and Intelligent Systems*, Lecture Notes in Artificial Intelligence, Vol. 6752, pages 31–40. Springer-Verlag Berlin Heidelberg, June 2011.

[47] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the First International Conference on Autonomous Agents*, Agents '97, pages 340–347, New York, NY, USA, 1997. ACM.

[48] R.E. Korf. A simple solution to pursuit games. In *Eleventh International Workshop on Distributed Artificial Intelligence*, pages 183–194, Glen Arbor, Michigan, USA, February 1992.

[49] M. Latek, R. Axtell, and B. Kaminski. Bounded rationality via recursion. In *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, volume 1, pages 457–464. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[50] V. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11(1), January 1999.

[51] R. Levy and J. Rosenschein. *A Game Theoretic Approach to Distributed Artificial Intelligence and the Pursuit Problem*, pages 129–146. Elsevier Science Publishers B.V., 1992.

[52] C.K. Lin, V.I. Zadorozhny, P.V. Krishnamurthy, H.H. Park, and C.G. Lee. A distributed and scalable time slot allocation protocol for wireless sensor networks. *IEEE Transactions on Mobile Computing*, 10(4):505–518, April 2011.

[53] M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *In Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann, 1994.

[54] J.R. Marden, G. Arslan, and J.S. Shamma. Cooperative control and potential games. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 39(6):1393 –1407, December 2009.

[55] T.A. Montgomery and E.H. Durfee. Using mice to study intelligent dynamic coordination. In *Proceedings of the Second International Conference on Tools for Artificial Intelligence*, pages 438–444. IEEE, November 1989.

[56] J. Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35(4):435–438, April 1987.

[57] M. Nair, R. Tambe and S. Marsella. Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Autonomous Agents and Multi-Agent Systems*, pages 552–559, 2003.

[58] R. Nair and M. Tambe. Hybrid bdi-pomdp framework for multiagent teaming. *Journal of Artificial Intelligence Research*, 23:367–420, April 2005.

[59] J.F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.

[60] J.F. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[61] N. Nie and C. Comaniciu. Adaptive channel allocation spectrum etiquette for cognitive radio networks. In *New Frontiers in Dynamic Spectrum Access Networks, 2005. DySPAN 2005. 2005 First IEEE International Symposium on*, pages 269–278, November 2005.

[62] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12(2/3):233–250, 1998.

[63] M.J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[64] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005.

[65] L.E. Parker. Alliance: an architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, April 1998.

[66] L.E. Parker and B.A. Emmons. Cooperative multi-robot observation of multiple moving targets. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2082–2089, 1997.

[67] T. Parsons. Pursuit-evasion in a graph. In Yousef Alavi and Don Lick, editors, *Theory and Applications of Graphs*, volume 642 of *Lecture Notes in Mathematics*, pages 426–441. Springer Berlin / Heidelberg, 1978.

[68] I. Rhee, A. Warrier, J. Min, and L. Xu. Drand: Distributed randomized tdma scheduling for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 8(10):1384–1396, October 2009.

[69] L.G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5:28–42, April 1975.

[70] J. Robinson. An iterative method of solving a game. *The Annals of Mathematics*, 54(2): 296–301, September 1951.

[71] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory Series B*, 58:22–33, May 1993.

[72] J.S. Shamma and G. Arslan. Dynamic fictitious play, dynamic gradient play, and distributed convergence to nash equilibria. *Automatic Control, IEEE Transactions on*, 50(3): 312–327, March 2005.

[73] Y. Shoham and K. Keyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.* Cambridge University Press, Cambridge, U.K., 2009.

[74] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171, 2007.

[75] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *In Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 541–548, 2000.

[76] K. Skrzypczyk. Game theory based target following by a team of robots. In *Robot Motion and Control, 2004. RoMoCo'04. Proceedings of the Fourth International Workshop on*, pages 91–96, June 2004.

[77] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[78] P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer.* MIT Press, 2000.

[79] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, June 1999.

[80] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, July 2000.

[81] P. Stone, G.A. Kaminka, S. Kraus, and J.S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence*, July 2010.

[82] P. Stone, G.A. Kaminka, and J.S. Rosenschein. Leading a best-response teammate in an ad hoc team. In Esther David, Enrico Gerding, David Sarne, and Onn Shehory, editors, *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 132–146. November 2010.

[83] D. Suryadi and P.J. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *In Proceedings of the Seventh International Conference on User Modeling*, pages 223–232, 1999.

[84] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[85] C. Undeger and F. Polat. Multi-agent real-time pursuit. *Autonomous Agents and Multi-Agent Systems*, 21:69–107, 2010.

[86] V.A. Utgoff and R.L. Kashyap. A pursuit and evasion problem with measurement uncertainty. *Journal of Optimization Theory and Applications*, 6(1):68–88, 1970.

[87] N. Vaidya, A. Dugar, S. Gupta, and P. Bahl. Distributed fair scheduling in a wireless lan. 4(6):616–629, November 2005.

[88] J.M. Vidal and E.H. Durfee. Recursive agent modeling using limited rationality. *First International Conference on Multi-Agent Systems (ICMAS)*, pages 376–383, June 1995.

[89] J.M. Vidal and E.H. Durfee. The moving target function problem in multi-agent learning. In *In Proceedings of the International Conference on Multi Agent Systems*, pages 317–324, July 1998.

[90] R. Vidal, O. Shakernia, H.J. Kim, D.H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18(5):662–669, October 2002.

[91] J. Von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[92] H. Wang, C. Shen, and K.G. Shin. Adaptive-weighted packet scheduling for premium service. In *Communications, 2001. IEEE International Conference on*, volume 6, pages 1846–1850, 2001.

[93] G. Weib. Learning to coordinate actions in multi-agent systems. In *In Proceedings of the Thirteenth Interational Conference on Artificial Intelligence*, pages 311–316. Morgan Kaufmann, 1993.

[94] M. Weinberg and J.S. Rosenschein. Best-response multiagent learning in non-stationary environments. In *In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 2.

[95] G. Welch and G. Bishop. An introduction to the kalman filter, July 2006.

[96] M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10:115–152, 1995.

[97] F. Wu, S. Zilberstein, and X. Chen. Online planning for ad hoc autonomous agent teams. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, Barcelona, Spain, 2011.

[98] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 616–623. ACM Press, 2001.

[99] H. Yanco and L.A. Stein. An adaptive communication protocol for cooperating mobile robots. In *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 478–485. MIT Press, 1993.

[100] L.S. Zettlemoyer, B. Milch, and L.P. Kaelbling. Multi-agent filtering with infinitely nested beliefs. *Neural Information Processing Systems*, 2008.

[101] C. Zhang, S. Abdallah, and V. Lesser. Integrating organizational control into multi-agent learning. In Sichman Decker and Castelfranchi Sierra, editors, *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems*, volume 2, pages 757–764, Budapest, Hungary, 2009.

Vita

Kyle Hollins Wray

Kyle Hollins Wray attended The Pennsylvania State University from 2005 to 2009, obtaining Bachelor of Science degrees both in Computer Science and Mathematics. He continued his education there from 2010 to 2012 as a Master of Science degree candidate in Computer Science and Engineering and a Master of Arts degree candidate in Mathematics, with a minor in Computational Science. His graduate work was supported by The Pennsylvania State University's Applied Research Laboratory, where he worked under Dr. Benjamin Thompson and Dr. Russ Burkhardt.

This thesis includes work done toward the following papers in preparation:

- Wray, Kyle H. and Thompson, Benjamin B. "The Defender Problem: An Investigation into Multi-Agent Learning in Information-Sparse Environments." Target: IEEE Transactions on Systems, Man, and Cybernetics, 2012-2013.
- Wray, Kyle H. and Thompson, Benjamin B. "An Ad Hoc Distributed Communication Architecture for Dynamic Multi-Agent Systems." Target: IEEE Transactions on Systems, Man, and Cybernetics, 2012-2013.
- Wray, Kyle H. and Thompson, Benjamin B. "Defender: Robotic Multiagent Coordination in Realistic Predator-Prey Scenarios" Target: IEEE Transactions on Robotics, 2013.
- Wray, Kyle H. and Thompson, Benjamin B. "Seeing the Unseen: Matrix Game Reconstruction for Highly Dynamic Multi-Agent Systems." Target: Journal of Autonomous Agents and Multi-Agent Systems, 2013.
- Wray, Kyle H. and Thompson, Benjamin B. "Role Distribution Games: A Mathematical Framework Solving the Problem of Cooperation in Highly Dynamic Scenarios." Target: Journal of Autonomous Agents and Multi-Agent Systems, 2013.