ANALYSIS OF SOFT FRIEND OR FOE REINFORCEMENT LEARNING ALGORITHM IN MULTIAGENT ENVIRONMENT

By

MICHAEL D. WIDENER

Bachelor of Science in Computer Science
Colorado School of Mines
Golden, Colorado, United States of America
2002

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2010

ANALYSIS OF SOFT FRIEND OR FOE REINFORCEMENT LEARNING ALGORITHM IN MULTIAGENT ENVIRONMENT

Thesis Approved:

Dr. Douglas Heisterkamp

Thesis Advisor

Dr. Blayne Mayfield

Dr. John P. Chandler

Dr. Mark E. Payton

Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to thank...

Dr. Doug Heisterkamp for suggested improvements to the algorithms analysed and environments used for analysis. Also for suggested improvements to the program implemented to run the algorithms and collect results; including the method used to convert state action value estimates into a policy. Recommendations on the notation and LateX functions were greatly appreciated.

Dr. Doug Heisterkamp, Dr. K.M. George, Dr. John Chandler for their time, patience, and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Two existing Multiagent Reinforcement Learning Algorithms are compared with a third algorithm named the Soft Friend or Foe algorithm. The Soft Friend or Foe algorithm introduces the use of correlation of returns into the Friend or Foe algorithm[1] in an attempt to gain the advantages of the Nash-Q Learning[2][3] algorithm without the added complexity of calculating Nash Equilibrium Points[4][5]. I have not come across any papers describing investigations of a technique similar to the Soft Friend or Foe algorithm. Littman's Friend or Foe[1] and Q-Learning[6] are the algorithms used to compare with and investigate the potential of the Soft Friend or Foe algorithm.

A review of important concepts in Off Policy Reinforcement Learning is provided which includes the general equation for Q-Learning[6] that all of the algorithms compared in this study use. The Friend or Foe and Nash-Q algorithms are described. An explanation of the Soft Friend or Foe algorithm follows with equations and pseudo code. The simple three by three grid environments that are used for comparison are described. The environments are all finite state games[6], for which the Markov decision property[6] holds. Finally the results of the comparison are summarized and analysed.

In the worst cases Soft Friend or Foe results in returns that are comparable to returns recieved using Q-Learning. In one case Soft Friend or Foe results in positive returns comparable to Friend or Foe returns while negative returns result from using the Q-Learning algorithm.

State action value estimates for the Soft Friend or Foe algorithm are shown to

consistently converge faster than state action value estimates of the other two algorithms. Policies generated from the state action value estimates show that the Soft Friend or Foe algorithm is more aggressive than the Q-Learning algorithm in the environments used. The Q-Learning algorithm executed in half the time of the Friend or Foe and Soft Friend or Foe algorithms.

CHAPTER 2

NOTATION

- **In following notation $\times$ is used for Cartesian product and $*$ is used for multiplication**

- **Environment is described by a tuple** $\{\mathcal{S}, \boldsymbol{X}, \mathcal{A}, \mathcal{F}, \delta_A, \boldsymbol{\delta_p}, f_p, \boldsymbol{\delta_r}\}$

- $\mathcal{R}$ denotes the set of *real numbers*

- $\mathcal{S}$ = set of states $\{s_1, s_2, s_3, \ldots s_m\}$

- $\boldsymbol{X}$ = sequence of agents $(x_1, x_2, x_3, \ldots x_n)$

- $\mathcal{A}$ = set of actions $\{a_1, a_2, a_3, \ldots a_l\}$

- $\mathcal{F}$ = set of reward functions $\{f_1, f_2, f_3, ...f_q\}$
  which take no input and return a real number

- $\delta_A(x, s) = \mathcal{D} | x \in \boldsymbol{X}, s \in \mathcal{S}, \mathcal{D} \subseteq \mathcal{A}$

- Set of actions available to agent x at state s

- $\mathbf{K}(s) = \delta_A(x_1, s) \times \delta_A(x_2, s) \times \ldots \delta_A(x_n, s)$

- Matrix of possible actions at state s. A row in the matrix is a possible action sequence or vector at state s. A column, c_i, in the matrix is a set of actions available to a single agent, x_i, at state s. An action sequence is a sequence of actions where every agent contributes one action. The set of possible action sequences is the Cartesian product of the actions available to each agent at state

s. The ith element of the action sequence is an action available to the ith agent of the agent sequence $\boldsymbol{X}$ at state s.

- $\boldsymbol{\Lambda}(s) \in \mathbf{K}(s)$

- Action sequence: a sequence of actions where every agent contributes one action. The ith element of the action sequence is an action available to the ith agent of the agent sequence $\boldsymbol{X}$ at state s.

- $\boldsymbol{\delta_p}(s, \boldsymbol{\Lambda}(s), s') = p | s \in \mathcal{S} \wedge s' \in \mathcal{S}$,
  p *is a real number in range* $[0,1] \wedge \sum_{s' \in \mathcal{S}} \boldsymbol{\delta_p}(s, \boldsymbol{\Lambda}(s), s') = 1$

  $\mathcal{P} \subset \mathcal{R} | p \in \mathcal{P} \rightarrow p \geq 0 \wedge p \leq 1$

  $\boldsymbol{\delta_p} : \mathcal{S} \times \mathcal{A}^n \times \mathcal{S} \rightarrow \mathcal{P}$

- Transition probability matrix defines the probability that action vector $\boldsymbol{\Lambda}(s)$ will result in a transition from state s to next state s'.

- $f_p(s, \boldsymbol{\Lambda}(s)) = s' | s' \in \mathcal{S}$, the probability that
  f_p returns s' is equal to $\boldsymbol{\delta_p}(s, \boldsymbol{\Lambda}(s), s')$

  $p : \mathcal{S} \times \mathcal{A}^n \rightarrow \mathcal{S}$

- Transition function accepts the current state, s, and an action vector, $\boldsymbol{\Lambda}(s)$, and returns the next state s'.

- $\boldsymbol{\delta_r}(x, \boldsymbol{\Lambda}(s), s, s') = f | x \in \boldsymbol{X}, s \in \mathcal{S}, s' \in \mathcal{S}, f \in \mathcal{F}$

  $\delta_r : \boldsymbol{X} \times \mathcal{A}^n \times \mathcal{S}^2 \rightarrow \mathcal{F}$

- Reward function matrix defines the reward function that will be used for a given agent, x, for a given transition from state s to next state s' on action vector $\boldsymbol{\Lambda}(s)$. State s may be equal to s'.

  Behavior of agents is described by policy, $\boldsymbol{\pi}$

- $\boldsymbol{\pi}(x,s) = (\pi(x,s,a_1), \pi(x,s,a_2) \ldots \pi(x,s,a_l)) \,|x \in \boldsymbol{X}, s \in \mathcal{S},$
  $\pi(x,s,a)$ gives a value in range $[0,1] \wedge \sum_{a \in \delta_A(x,s)} \pi(x,s,a) = 1$
  $\boldsymbol{\pi} : \boldsymbol{X} \times \mathcal{S} \rightarrow \mathcal{R}^l$
  $\pi : \boldsymbol{X} \times \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{P}$

- $\boldsymbol{\pi}(x,s)$ provides a probability distribution over elements of action sequence $\boldsymbol{\Lambda}(s)$

- $\boldsymbol{\Pi}(s) = \boldsymbol{\pi}(x_1,s) \times \boldsymbol{\pi}(x_2,s) \times \ldots \boldsymbol{\pi}(x_n,s)$

- $\boldsymbol{\Pi}(s)$ provides a probability distribution over all possible action sequences at state s, $\mathbf{K}(s)$. So given an action sequence $\boldsymbol{\Lambda}(s)$, $\boldsymbol{\Pi}(s)$ defines the probability of that action sequence which is equal to the product of the probability of each action in the action sequence at state s.

- $a(x_i, \boldsymbol{\Lambda}(s)) = a_i | a_i \in \delta_A(x_i,s) \wedge \boldsymbol{\Lambda}(s) \in \delta_A(x_1,s) \times \delta_A(x_2,s) \times \ldots \delta_A(x_{i-1},s) \times a_i \times \delta_A(x_{i+1},s) \times \ldots \delta_A(x_n,s)$

- The action from agent x_i in the action sequence $\boldsymbol{\Lambda}(s)$. If x_i is the ith element in the agent sequence $\boldsymbol{X}$, then a_i is the ith element of the action sequence $\boldsymbol{\Lambda}(s)$.

- $\pi(\boldsymbol{\Lambda}(s), s) = \pi(x_1, s, a(x_1, \boldsymbol{\Lambda}(s))) * \pi(x_2, s, a(x_2, \boldsymbol{\Lambda}(s))) \ldots \pi(x_n, s, a(x_n, \boldsymbol{\Lambda}(s)))$

- The probability defined in $\boldsymbol{\Pi}(s)$ for action sequence $\boldsymbol{\Lambda}(s)$

- **Agents receive feedback through a sequence of return values, R**

- $R(x) = (r_1, r_2, \ldots r_i \ldots r_t) \,| x \in \boldsymbol{X},\ \forall i\ r_i \in f \wedge f = \boldsymbol{\delta_r}(x, \boldsymbol{\Lambda}(s), s, s')$
  $\wedge\, s' = f_p(s, \boldsymbol{\Lambda}(s), \boldsymbol{\delta_p})\ for\ all\ s \in \mathcal{S}$

- Sequence of returns for agent x.

- $R(x,s) \subseteq R(x) \,| s \in \mathcal{S}, r_i \in R(x,s) \Leftrightarrow r_i \in f \wedge f = \boldsymbol{\delta_r}(x, \boldsymbol{\Lambda}(s), s, s')$
  $\wedge\, s' = f_p(s, \boldsymbol{\Lambda}(s), \boldsymbol{\delta_p}) \wedge r_i \in R(x)$

- Sequence of returns for agent x when transitioning from state s to some other state.

- $R(x,s,a) \subseteq R(x,s) \,|a \in \delta_A(x,s), r_i \in R(x,s,a) \Leftrightarrow$
  $r_i \in f \wedge f = \boldsymbol{\delta_r}(x, \boldsymbol{\Lambda}(s), s, s') \wedge s' = f_p(s, \boldsymbol{\Lambda}(s), \boldsymbol{\delta_p}) \wedge$
  $\boldsymbol{\Lambda}(s) \in \delta_A(x_1,s) \times \delta_A(x_2,s) \ldots \times a \times \ldots \delta_A(x_n,s) \wedge r_i \in R(x,s)$

- Sequence of returns for agent x when transitioning from state s to some other state and action a selected for agent x.

- $R(x,s,\boldsymbol{\Lambda}(s)) \subseteq R(x,s) \,|r_i \in R(x,s,\boldsymbol{\Lambda}(s)) \Leftrightarrow r_i \in f \wedge$
  $f = \boldsymbol{\delta_r}(x, \boldsymbol{\Lambda}(s), s, s') \wedge s' = f_p(s, \boldsymbol{\Lambda}(s), \boldsymbol{\delta_p}) \wedge r_i \in R(x,s)$

- Sequence of returns for agent x when transitioning from state s to some other state on action sequence $\boldsymbol{\Lambda}(s)$.

- $r_j(x,s,a) = r_j | R(x,s,a) = (r_1, r_2, \ldots r_i, r_j \ldots r_k) \wedge |(r_1 \ldots r_i)| = j-1$

- jth element of $R(x,s,a)$

- $r_j(x,s,\boldsymbol{\Lambda}(s)) = r_j | R(x,s,\boldsymbol{\Lambda}(s)) = (r_1, r_2, \ldots r_i, r_j \ldots r_k) \wedge |(r_1 \ldots r_i)| = j-1$

- jth element of $R(x,s,\boldsymbol{\Lambda}(s))$

- $r_j(x,s) = r_j | R(x,s) = (r_1, r_2, \ldots r_i, r_j \ldots r_k) \wedge |(r_1 \ldots r_i)| = j-1$

- jth element of $R(x,s)$

- **Each return in R has a corresponding state in the state history, H**

- $H = (h_1, h_2, \ldots h_i \ldots h_t) \,|for\ i > 1\ h_i \in f_p(h_{i-1}, \boldsymbol{\Lambda}(h_{i-1}), \boldsymbol{\delta_p}), h_1 \in \mathcal{S}$

- Sequence of states

- $H(s) \subseteq H | s \in \mathcal{S}, h_i \in H(s) \rightarrow h_i \in f_p(s, \boldsymbol{\Lambda}(s), \boldsymbol{\delta_p})$

- Sequence of next states when transitioning from state s

- $H(x,s,a) \subseteq H(s) \,|a = a(x, \mathbf{\Lambda}(s)), h_i \in H(x,s,a) \rightarrow h_i \in f_p(s, \mathbf{\Lambda}(s), \boldsymbol{\delta_p})$

- Sequence of next states when transitioning from state s and action a selected for agent x

- $H(s, \mathbf{\Lambda}(s)) \subseteq H(s) \,|h_i \in H(s, \mathbf{\Lambda}(s)) \rightarrow h_i \in f_p(s, \mathbf{\Lambda}(s), \boldsymbol{\delta_p})$

- Sequence of next states when transitioning from state s on action sequence $\mathbf{\Lambda}(s)$

- $h_j(x,s,a) = h_j | H(x,s,a) = (h_1, h_2, \ldots h_i, h_j \ldots h_k) \wedge |(h_1 \ldots h_i)| = j-1$

- jth element of $H(x,s,a)$

- $h_j(s, \mathbf{\Lambda}(s)) = h_j | H(s, \mathbf{\Lambda}(s)) = (h_1, h_2, \ldots h_i, h_j \ldots h_k) \wedge |(h_1 \ldots h_i)| = j-1$

- jth element of $H(s, \mathbf{\Lambda}(s))$

- $h_j(s) = h_j | H(s) = (h_1, h_2, \ldots h_i, h_j \ldots h_k) \wedge |(h_1 \ldots h_i)| = j-1$

- jth element of $H(s)$

2.0.1 Notation for Algorithms

- **Subsets of $\mathbf{K}(s)$**

- $\mathbf{K}(\boldsymbol{B}, s) = \delta_A(x_1, s) \times \delta_A(x_2, s) \times \ldots \delta_A(x_b, s) \,|\boldsymbol{B} \subseteq \boldsymbol{X} \wedge b = |\boldsymbol{B}| \wedge \forall x_i \in x_1..x_b \; x_i \in \boldsymbol{B}$

- Set of possible action sequences at state s where only agents in subsequence $\boldsymbol{B}$ contribute actions. The ith element of the action sequence is an action available to the ith agent of the agent sequence $\boldsymbol{B}$ at state s.

- $\mathbf{\Lambda}(\boldsymbol{B}, s) \in \mathbf{K}(\boldsymbol{B}, s)$

- Limited action sequence where only agents in subsequence $\boldsymbol{B}$ constribute an action. The ith element of the action sequence is an action available to the ith agent of the agent sequence $\boldsymbol{B}$ at state s.

- $a\left(x_i, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) = a_i | a_i \in \delta_A\left(x_i, s\right) \wedge \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right) \in \delta_A\left(x_1, s\right) \times \delta_A\left(x_2, s\right) \times \ldots \delta_A\left(x_{i-1}, s\right) \times a_i \times \delta_A\left(x_{i+1}, s\right) \times \ldots \delta_A\left(x_b, s\right)$

- The action from agent x_i in the action sequence $\boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)$. If x_i is the ith element in the agent sequence $\boldsymbol{B}$, then a_i is the ith element of the action sequence $\boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)$.

- $\Phi\left(x_i, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) = UV | U = \left(a\left(x_1, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right), a\left(x_2, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right), \ldots a\left(x_{i-1}, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right)\right) \wedge V = \left(a\left(x_{i+1}, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right), \ldots a\left(x_b, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right)\right) \wedge |UV| = |\boldsymbol{B}| - 1$

- Action sequence resulting from removing an action from $\boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)$. The action removed is the action provided by agent x_i.

- $\Psi\left(x_i, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) = a\left(x_1, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) \times a\left(x_2, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) \times \ldots a\left(x_{i-1}, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) \times \delta_A\left(x_i, s\right) \times a\left(x_{i+1}, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) \times \ldots a\left(x_b, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) | x_i \in \boldsymbol{B} \wedge \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right) \in \Psi\left(x_i, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) \wedge |\boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)| = |\boldsymbol{B}|$

- Set of action sequences resulting from varying the action provided by agent x_i.

- $\psi\left(x_i, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) = \delta_A\left(x_1, s\right) \times \delta_A\left(x_2, s\right) \times \ldots \delta_A\left(x_{i-1}, s\right) \times a\left(x_i, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) \times \delta_A\left(x_{i+1}, s\right) \times \ldots \delta_A\left(x_b, s\right) | \forall i \in [1, b] \, x_i \in \boldsymbol{B} \wedge \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right) \in \psi\left(x_i, \boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)\right) \wedge |\boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)| = |\boldsymbol{B}|$

- Set of action sequences resulting from varying all of the actions in action sequence $\boldsymbol{\Lambda}\left(\boldsymbol{B}, s\right)$, but keeping action provided by agent x_i fixed.

- **Reward statistics**

- $\bar{r_k}\left(x, s, a\right)$ is the mean of the first k returns in the sequence $R\left(x, s, a\right)$

- $\bar{r_k}(x, s)$ is the mean of the first k returns in the sequence $R(x, s)$
- $\bar{R_k}(x, s, \mathbf{\Lambda}(s))$ is the mean of the first k returns in the sequence $R(x, s, \mathbf{\Lambda}(s))$
- $Var_k(x, s)$ is the variance of the first k returns in the sequence $R(x, s)$
- $\sigma_k(x_1, x_2, s)$ is the covariance between the first k returns of sequences $R(x_1, s)$ and $R(x_2, s) | x_1 \in \boldsymbol{X}, x_2 \in \boldsymbol{X}$
- $\rho_k(x_1, x_2, s)$ is the correlation between the first k returns of sequences $R(x_1, s)$ and $R(x_2, s) | x_1 \in \boldsymbol{X}, x_2 \in \boldsymbol{X}$
- **Functions**
- $Q(x, s, \mathbf{\Lambda}(\boldsymbol{B}, s))$ is the expected return for agent $x \in X$ when taking action sequence $\mathbf{\Lambda}(\boldsymbol{B}, s)$ at state $s \in \mathcal{S}$ where $a \in \delta_A(x, s) \wedge a \in \mathbf{\Lambda}(\boldsymbol{B}, s)$
- $V(x, s)$ is the value approximation for state $s \in \mathcal{S}$ with respect to agent $x \in \boldsymbol{X}$
- γ is a constant discount factor in range $[0, 1]$
- $\alpha(k)$ is a function of k used to dampen the change in Q

CHAPTER 3

LITERATURE REVIEW

3.1 Sutton and Barto Reinforcement Learning

The basics of Reinforcement Learning are described in *Sutton and Barto's book, Reinforcement Learning: An Introduction* [6].

The key to this branch of Artificial Inteligence is estimating the value of each state-action pair in the environment. A sequence of returns generated by a state-action pair is the bases for estimating the relative goodness of that state action pair.

One method presented in Sutton and Barto's book is to take the average of the returns as a reasonable estimate of the value of that state-action pair[6]. There are other ways to estimate the expected value of a state-action pair, but all of the algorithms I am considering will utilize this simple statistical method. Equation 3.1 [6, p. 27]:

$$Q(x,s,a) = \frac{1}{k}\left(\sum_{r_j \in R(x,s,a)} r_j\right) \tag{3.1}$$

- $k = |R(x,s,a)|$

The equation above only accounts for the immediate returns from an action. If there are multiple states in the environment, then relying solely on equation 3.1 will isolate the states and relay no feedback from future consequences of their actions. More information is needed.

The link that feeds information from state to state is the approximate value of the next state. $V(x,s)$ is the notation used for an approximation of the value of state s for agent x. The three algorithms that I wish to compare each calculate

$V(x,s)$ differently in an attempt to approximate a state value that would result from an optimal policy. The addition of $V(x,s)$ to equation 3.1 results in the following equation. Equation 3.3 [6, p. 75]:

$$V(x,s) \approx \max_{\mathbf{\Pi}} V^{\mathbf{\Pi}}(x,s) \tag{3.2}$$

$$Q(x,s,a) = \frac{1}{k}\left(\sum_{r_j \in R(x,s,a)} [r_j + V(x,s_j)]\right) \tag{3.3}$$

- $k = |R(x,s,a)|$
- $r_j = r_j(x,s,a)$
- $s_j = h_j(x,s,a)$

To avoid entering into infinite loops, and to add some incentive to reach a final state more quickly (win/loss), a discount factor, $\gamma \in [0,1]$, is added to the state action value equation. Equation 3.4 [6, p. 75]:

$$Q(x,s,a) = \frac{1}{k}\left(\sum_{r_j \in R(x,s,a)} [r_j + \gamma V(x,s_j)]\right) \tag{3.4}$$

- $k = |R(x,s,a)|$
- $r_j = r_j(x,s,a)$
- $s_j = h_j(x,s,a)$

A discount factor close to one means that a long term goal is more acceptable than with a discount factor close to zero[6].

It is easier to store a previous state-action value and a counter, k, than it is to store a set of returns. Also it would be nice to update the state-action value estimate after each return. With this in mind, observe that the equation above can be re-written into an iterative form. Equation 3.5 [6, p. 148]:

$$Q(x,s,a) \leftarrow Q(x,s,a) + \frac{1}{k}[r_k + \gamma V(x,s_{k+1}) - Q(x,s,a)] \tag{3.5}$$

- $k = |R(x, s, a)|$
- $r_k = r_k(x, s, a)$
- $s_{k+1} = h_{k+1}(x, s, a)$

One final tweak is the replacement of $\frac{1}{k}$ with the function $\alpha(k)$. The motivation for this is to allow the state-action value estimate to adjust to changes in the environment. Alpha may decrease rapidly until a minimum value is reached which allows the value of Q(s,a) to change where $\frac{1}{k}$ approaches zero; preventing an update to the estimate. Switching $\alpha(k)$ for $\frac{1}{k}$ gives the more general equation 3.6 [6, p. 148]:

$$Q(x, s, a) \leftarrow Q(x, s, a) + \alpha(k)\left[r_k + \gamma V(x, s_{k+1}) - Q(x, s, a)\right] \tag{3.6}$$

- $\alpha(k)$ may be constant
- $k = |R(x, s, a)|$
- $r_k = r_k(x, s, a)$
- $s_{k+1} = h_{k+1}(x, s, a)$

3.2 Q-Learning

Q-Learning uses the maximum expected return value of the next state to approximate the value of the next state[6, p. 148]. Equation 3.7 [6]:

$$Q(x, s, a) \leftarrow Q(x, s, a) + \alpha(k)\left[r_k(x, s, a) + \gamma \max_b Q(x, s_{k+1}, b) - Q(x, s, a)\right] \tag{3.7}$$

- $k = |R(x, s, a)|$
- $\alpha(k)$ may be constant
- $r_k = r_k(x, s, a)$

- $s_{k+1} = h_{k+1}(x, s, a)$

Like all of the algorithms I will be comparing, Q-Learning is an off-policy learning algorithm[6]. Off-Policy reinforcement learning algorithms are those algorithms which provide state-action value estimates that are independant of the policy being used to select actions[6]. An action must be selected enough to allow it's state action value estimate to converge to some value.

Q-Learning is proven to give state-action value estimates that converge to the optimal value in single agent environments[7]. I do not know if the same is true for environments with multiple agents.

The following algorithms will make use of knowledge of the actions taken by other agents in the environment. The returns that other agents receive are also known. Equation 3.6 will be used to calculate $Q(x, s, \mathbf{\Lambda}(s))$. Each of the following algorithms will generate a value for $V(x, s)$.

3.3 Littman's Friend or Foe

Littman suggested a simplified algorithm in which every other agent in the game is identified as a friend or a foe[1]. The Friend or Foe algorithm reduces the calculation of V(s) to a mini-max problem[1]. For example, say there are two agents in the environment: our principle agent, x_1, for which we are calculating state action values, and the accomplice/opponent, x_2. Agent x_1 produces action a_1, and agent x_2 produced action a_2. If x_2 were identified as a friend, then the *Friend* function [1] below would be used to calculate $V(x_1, s)$. If x_2 is a foe, then the value of $V(x_1, s)$ is given by the *Foe* equation [1].

- $$Friend(x_1, x_2, s) = \max_{a_1, a_2} [Q(x_1, s, (a_1, a_2))] \tag{3.8}$$

•

$$Foe\left(x_1, x_2, s\right) = \max_{a_1} \min_{a_2} \left[Q\left(x_1, s, \left(a_1, a_2\right)\right)\right] \tag{3.9}$$

The general equation is a typcial min max equation where actions are chosen for foes that minimize the best expected return that can be achieved by the primary agent. Actions are then selected for friends and the primary agent that maximize the expected return of the primary agent. Here is the more general equation where x_1 is the principle agent. agents in the set $U = \{x_2, ...x_u\}$ are friends, and agents in the set $V = \{x_{u+1}, x_{u+2}, ...x_{u+v}\}$ are foes. Equation 3.10 [1]:

$$V\left(x_1, s\right) = \max_{\mathbf{\Lambda}(U,s)} \min_{\mathbf{\Lambda}(V,s)} \left[Q\left(x_1, s, \mathbf{\Lambda}\left(U, s\right) \mathbf{\Lambda}\left(V, s\right)\right)\right] \tag{3.10}$$

Friend or Foe is best viewed as a benchmark. The explicit classification of the secondary agents as either a friend or a foe means that the algorithm is not as self sufficient as Q-Learning. The advantages of Friend or Foe are that the state action value estimates always converge and it is less expensive computationally[1].

3.4 Hu and Wellman's Nash-Q

Nash-Q[2] is a multiagent reinforcement learning algorithm that attempts to find an estimate for the value of a state, V(x,s), that is close to optimal using the concept of an Equilibrium Point [5]. An estimate of V(x,s) that is optimal is the value that would result from following an optimal policy [6]. An optimal policy maximizes the returns for an agent [6]. Nash-Q was not implemented, but the algorithms is described here so that the motivation for the Soft Friend or Foe algorithm is more clear.

Hu and Wellman proposed that some sort of equilibrium value could be found among the competing/cooperating agents for each state, and that the equilibrium value should be used for the approximation of the state's value, $V(x, s)$ [2]. A Nash Equilibrium is the concept they used to find this equilibrium value [5].

A Nash Equilibrium is a set of policies among agents, $\mathbf{\Pi}$, such that no individual agent can improve their expected return by changing their policy, as long as the policies of other agents remain unchanged [2]. Here is an example to demonstrate: return values for agent a and agent b are shown in the figure 3.1.

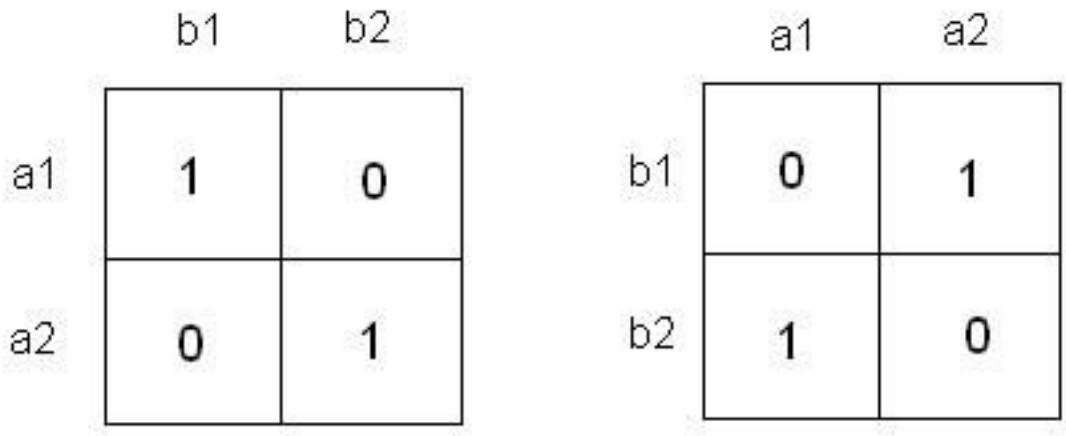

Figure 3.1: Bi matrix game

If agent a chooses action 1 and agent b chooses action 2, then agent a will get a return of zero and agent b will get a return of 1. The Nash Equilibrium for this example is for agent a to choose action 1 50% of the time and agent b to choose action 1 50% of the time. Bernhard Von Stengel provided a survey of methods that can be used to calculate a Nash Equilibrium[8].

$$V(x,s) \leftarrow \sum_{\mathbf{\Lambda}(s)\in\boldsymbol{\kappa}(s)} \left[Q\left(x,s,\mathbf{\Lambda}\left(s\right)\right) * \boldsymbol{\pi}\left(\mathbf{\Lambda}\left(s\right),s\right)\right] \tag{3.11}$$

- $\mathbf{\Pi}\left(s\right)$ is an equilibrium policy

The problem with the Nash-Q approach is that there may be more than one Nash Equilibrium for each n-matrix game, where n is the number and dimension of matrices [2]. The approximation of the state action value may never converge if a different $V(x,s)$, which is based on the Nash Equilibrium, is calculated each time. Another

problem is that the complexity of calculating an equilibrium point increases quickly with the number of agents in the game[4], and the number of actions available to each agent[8].

CHAPTER 4

THESIS

4.1 Soft Friend or Foe

Soft Friend or Foe is a modification to the Friend or Foe[1] algorithm that I don't believe has been tried before. The goal of the modification is to move $V(x, s)$ closer to the value that would result from an optimal policy without the problems that result from using the Nash-Q[3] algorithm.

The correlation between the returns of a primary agent and secondary agent can be used to classify the secondary agent as a friend or a foe. The primary agent being the agent we want to estimate state values for. The correlation in return values of the two agents can be used to infer a policy for the secondary agent. The state action values for the primary agent are estimated using the policy imposed on the secondary agent. Finally the state value estimate is taken to be the maximum state action value at that state.

Nash-Q should give better results because the policy imposed on both agents to calculate $V(x, s)$ is an equilibrium policy, $\mathbf{\Pi}(s)$, which means that neither agent can increase their expected return by unilaterally changing their policy, $\boldsymbol{\pi}(x, s)$. However, calculating an equilibrium point is not simple and there may be more than one equilibrium point at a given state [3][8]. Soft Friend or Foe simply estimates a Nash Equilibrium point by increasing the probability that agent x_2 will select an action that results in a higher expected return for agent x_1 if the returns for the two agents are correlated. Calculating correlation is much simpler than calculating an equilibrium point. The results are more stable because there is only one correlation value

for each pair of agents.

Friend or Foe is less computationally expensive because no information is obtained about a pair of agents besides a boolean indication of their relation, which is constant. Soft Friend or Foe is more complex than the original Friend or Foe algorithm, but does not require the programmer to identify each pair of agents as friends or foes. One simple way to identify another agent as a friend or a foe is to observe whether or not the actions that benefit the primary agent also benefit the other agent. If there is a strong positive correlation in returns, then the other agent can be treated as a friend. If there is a negative correlation in returns, then the other agent should be treated as an opponent. In the case of a weak correlation, it would be inappropriate to classify another agent as a friend or foe. Such classifications might cause severe fluctuations in the state value estimate, $V(x, s)$. A method is needed to soften the transition between friend and foe. A linear relation may work. The next obstacle is the calculation of the correlation which requires knowledge of ALL of the previous returns for both agents, as well as knowledge of the mean for both agents. Fortunately the mean can be approximated by the average incrementally. The covariance can also be calculated incrementally, and the two results can be used to find the correlation.

4.1.1 Calculating Correlation

For every agent in the game, other than the primary agent, a value for the correlation with the primary agent is maintained at every state. The correlation is based on the sequence of return values for the primary agent, x_1, and some other agent, x_2. To describe the equations we need to define k as the number of returns both agents have received at that state.

The average return for agent x at state s after k returns, $\bar{r_k}(x, s)$, can be calculated iteratively ...

$$\bar{r_{k+1}}(x,s) \leftarrow \frac{\bar{r_k}(x,s) * k + r_{k+1}(x,s)}{k+1} \tag{4.1}$$

$Var_k(x,s)$ is the variance of the return for agent x after k observations, and can be calculated iteratively ...

$$\begin{aligned} Var_{k+1}(x,s) \leftarrow & \frac{\left[Var_k(x,s) * k + (r_{k+1}(x,s) - \bar{r_k}(x,s))^2\right]}{k+1} \\ & - (\bar{r_{k+1}}(x,s) - \bar{r_k}(x,s))^2 \end{aligned} \tag{4.2}$$

$\sigma_k(x_1, x_2, s)$ is the covariance between agent x_1 and agent x_2 at state s after k observations, and can be calculated iteratively ...

$$\begin{aligned} \sigma_{k+1}(x_1,x_2,s) \leftarrow & \frac{\sigma_k(x_1,x_2,s) * k}{k+1} \\ & + \frac{(r_{k+1}(x_1,s) - \bar{r_k}(x_1,s)) * (r_{k+1}(x_2,s) - \bar{r_k}(x_2,s))}{k+1} \\ & - (\bar{r_{k+1}}(x_1,s) - \bar{r_k}(x_1,s)) * (\bar{r_{k+1}}(x_2,s) - \bar{r_k}(x_2,s)) \end{aligned} \tag{4.3}$$

$\rho_k(x_1, x_2, s)$ is the correlation between agent x_1 and agent x_2 at state s after k observations ...

$$\rho_k(x_1,x_2,s) = \frac{\sigma_k(x_1,x_2,s)}{\sqrt{Var_k(x_1,s) * Var_k(x_2,s)}} \tag{4.4}$$

The covariance value falls into the range [-1,1]; -1 meaning the two agents are foes and 1 meaning the two agents are friends.

4.1.2 Calculating State Value Estimate

The correlation in returns can be used as a measurement of the expected degree of cooperation between the two agents. The result of the function $\rho_k(x_1, x_2, s)$ gives a

real number in the range $[-1, 1]$ for state s. A value of 1 means the agents are fully cooperative at that state. A value of -1 means the agents are fully competetive.

A state value estimate, $V(x_1, s)$, can be calculated using the results of $\rho_k(x_1, x_i, s)$. For all state action sequence value estimates, $Q(x_1, s, \Lambda(s))$, of the state s, a weight is applied for each opposing agent, x_i. The weight is calculated using the following equation which gives a value in the range [0,1] ($W : \boldsymbol{X}^2 \times \mathcal{S} \times \mathcal{A}^n \rightarrow \mathcal{P}$).

$$W(x_1, x_2, s, \Lambda(s)) = \frac{1}{2} + \rho_k(x_1, x_2, s) * \frac{Q(x_1, s, \Lambda(s)) - med_i(x_1, x_2, s, \Lambda(s))}{max_i(x_1, x_2, s, \Lambda(s)) - min_i(x_1, x_2, s, \Lambda(s))} \tag{4.5}$$

$$max_i(x_1, x_2, s, \Lambda(s)) = \max_{w \in \Psi(x_2, \Lambda(s))} Q(x_1, s, w) \tag{4.6}$$

$$min_i(x_1, x_2, s, \Lambda(s)) = \min_{w \in \Psi(x_2, \Lambda(s))} Q(x_1, s, w) \tag{4.7}$$

$$med_i(x_1, x_2, s, \Lambda(s)) = \frac{max_i(x_1, x_2, s, \Lambda(s)) + min_i(x_1, x_2, s, \Lambda(s))}{2} \tag{4.8}$$

The algorithm used to convert the weights into a state value estimate is in Appendix N and also at the end of this section. The idea is to generate a policy for each agent using the weights. Say we are calculating the estimated value of a state, $V(x_1, s)$, for agent x_1 and $\boldsymbol{w}$ is an action vector for state s containing action b for agent x_2 ($\boldsymbol{w} \in \boldsymbol{K}(s) | a(x_2, \boldsymbol{w}) = b$). The probability of selecting action b is the proportion of the sum of weights where action b is selected to the sum of all weights for all possible action vectors at this state.

$$\pi(x_2, s, b) = \frac{\sum_{\boldsymbol{\Lambda(s)} \in \psi(x_2, \boldsymbol{w})} W(x_1, x_2, s, \boldsymbol{\Lambda(s)})}{\sum_{\boldsymbol{\Lambda(s)} \in \boldsymbol{K}(s)} W(x_1, x_2, s, \boldsymbol{\Lambda(s)})} \tag{4.9}$$

Once the policy is derived for all other agents, $V(x_1, s)$ can be calculated by taking the maximum of the expected state action values. Suppose $\boldsymbol{w}$ is an action vector for

state s containing action a for agent x_1 ($\boldsymbol{w} \in \boldsymbol{K}(s) | a(x_1, \boldsymbol{w}) = a$). Suppose policy for agent x_1 is to always select action a ($\pi(x_1, s, a) = 1$).

$$Q(s,a) = \sum_{\boldsymbol{\Lambda}(s) \in \psi(x_1, \boldsymbol{w})} (\pi(\boldsymbol{\Lambda}(s), s) * Q(s, \boldsymbol{\Lambda}(\boldsymbol{s}))) \tag{4.10}$$

Now that a state action value is available for agent x_1 at state s, the Q-Learning algorithm can be used to finish determining the value of the state by selecting the best choice for agent x_1.

$$V(x_1, s) = \max_{a \in \delta_A(x_1, s)} Q(s,a) \tag{4.11}$$

The algorithm described in appendix N tries to minimize the complexity of calculating $V(x,s)$ by calculating intermediate $Q(x, \boldsymbol{B})$ values.

1. $B \leftarrow X$
2. while $|B| > 1$
   (a) $y \leftarrow x_i | \min_{x_i \in B} \rho_k(x, x_i, s)$
   (b) for all $w \in \psi(y, \Lambda(B, s))$
      i. $w' \leftarrow \Phi(y, w)$
      ii. $Q(x, s, w') \leftarrow \frac{\sum_{\Lambda(s) \in \Psi(y,w)} W(x,y,s,\Lambda(s)) * Q(x,s,\Lambda(s))}{\sum_{\Lambda(s) \in \Psi(y,w)} W(x,y,s,\Lambda(s))}$
   (c) $B \leftarrow B \cap \overline{y}$
3. $V(x,s) \leftarrow max_{w \in \Lambda(B,s)} Q(x,s,w)$

*$W(x, y, s, \Lambda(s))$ defined in equation 4.5

The Soft Friend or Foe algorithm starts by setting the agent sequence $\boldsymbol{B}$ equal to the agent sequence of the problem definition, $\boldsymbol{X}$. An agent, y, is selected from $\boldsymbol{B}$ that has the lowest correlation with the primary agent, x. For every available action sequence, $w' \in \boldsymbol{\Lambda}(\boldsymbol{B} \cap \overline{\boldsymbol{y}})$, at state s using actions from agent sequence $\boldsymbol{B}$ where the

action provided by agent y is omited, a new state action sequence value estimate, $Q(x, s, w')$, is calculated. The new state action sequence value estimate is calculated by weighting existing state action sequence value estimates.

A weight is determined by what happens when the action provided by agent y is varied. All other actions are held constant so we can imagine we are dealing with state action value estimates where the action is provided by agent y. The current action for y results in a state action value estimate that lies somewhere in the range of values that result by varying the action provided by agent y. Equation 4.5 shows that if the state action value estimate is greater than the value half way between the upper bound and lower bound, and the correlation is positive, then the weight will be greater. A negative correlation would mean that an estimated value that is closer to the upper bound would be given a low weighting.

The weights are used to calculate an expected value for $Q(x, s, w')$ that can be used in subsequent iterations until only $Q(x, s, a)$ remains where action a is the action provided by agent x. At that point the maximum state action value estimate is returned as the estimated value for the state.

CHAPTER 5

EXPERIMENTAL RESULTS AND ANALYSIS

5.1 Experimental Methodology

5.1.1 General Procedure

Agents have full knowlege of the environment state. Agents follow set policies in which actions are selected randomly; $\forall a \in \delta_A(x,s), \pi(x,s,a) = \frac{1}{|\delta_A(x,s)|}$. Agents recieve rewards and are aware of the rewards that other agents recieve.

Actions are selected randomly for each agent, with each action having an equal probability of being selected. This gives meaningful results because all of the algorithms I wish to compare are Off-Policy algorithms[6]. The next state is determined by the environment based on the actions taken by each player and environmental factors. The immediate reward function for each agent is determined by the state transition and the actions taken by each player. The reward function may be constant. The immediate reward is recorded for each agent for the original environment state and the set of actions taken.

A set number of episodes is executed for learning. The number of episodes is selected so that the state action sequence value estimates converge to a stable value. The same algorithm is used for both agents. A policy is then derived from the state actione sequence value estimates using the softmax equation[6, p. 30] with a temperature of 4.1. An equal number of episodes is run and the rewards for each agent for each episode are recorded. The number of iterations for each episode is also recorded.

5.1.2 Statistics To Be Collected

State action value estimates are monitored for convergence. Average execution time for calculating $V(x, s)$ is monitored for each algorithm. Policies are derived for each agent using the final state action value estimates. Average returns based on the derived policies and knowledge of the environment are reported for each agent. Key state-action value estimates are reported and analyzed for some games.

5.1.3 Games

The environment model is a simple grid world game where two agents compete to reach the final state as fast as possible without running into eachother[3]. The advantage of this environment is that the results are easily interpreted. Figure 5.1, summerizes a set of games described in Greenwald and Hall's paper [9], which are similar to games used in Hu and Wellman's 98 paper *Multiagent Reinforcement Learning: Theoretical Framework and Algorithm* [2] and 03 paper *Nash Q-Learning for General-Sum Stochasic Games* [3]. In grid game 1, the players, represented by stick figure people, receive one hundred points for reaching the goal, represented by a doorway. If the players attempt to move into the same square, each player receives a reward of -1. The game ends when one or more players have reached the goal. It is to the player's advantage to reach the goal as soon as possible so that they do not miss out on any points. Final states are states in which one or both players have reached the goal. Grid Game 2 has a shared goal and players are awarded 120 pts if they both enter the goal from the side. If one player enters the goal from the side while the other enters the goal from below, then the agent entering from below receives 125 points and the agent entering from the side receives 100 points. In Grid Game 3 there is one shared goal and agents receive 100 points for entering that goal, but there are two barriers that agents have a 50 percent chance of crossing on every attempt.

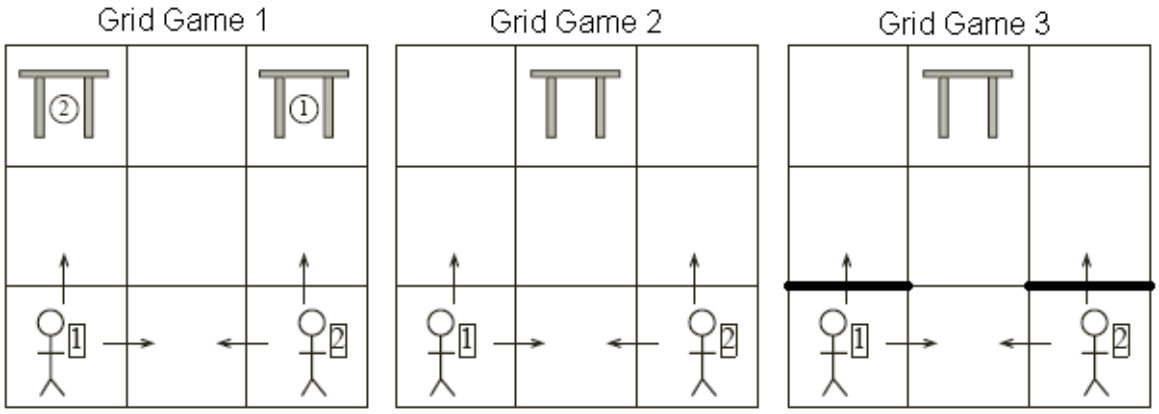

Figure 5.1: Grid games

5.2 Results

5.2.1 Returns

In the simple environments used, Soft Friend and Foe received returns comparable to the other two algorithms used. In the only test case where returns varied significantly, the Q-Learning algorithm resulted in returns that were negative on average while the value of returns provided by the other two algorithms (Figure 5.2) were positive. Soft Friend or Foe outperformed the other algorithms in head to head competition. However the difference in average returns per episode was well within one standard deviation in most cases.

The return for agent 1 over an episode is calculated by summing agent 1's returns for all iterations from the initial state until the final state. The same is done for agent 2. For each episode the return over that episode for agent 1 and agent 2 are added to give the return for that episode. The average return is calculated by taking the average of the returns for all episodes.

Figure E.1 shows the number of episodes with a return greater than or equal to x. The x axis value starts with the maximum reward possible and progresses toward the minimun. It is desirable for the line to reach the total number of episodes as quickly as possible. In grid game 1 the Soft Friend or Foe algorithm found the optimal solution

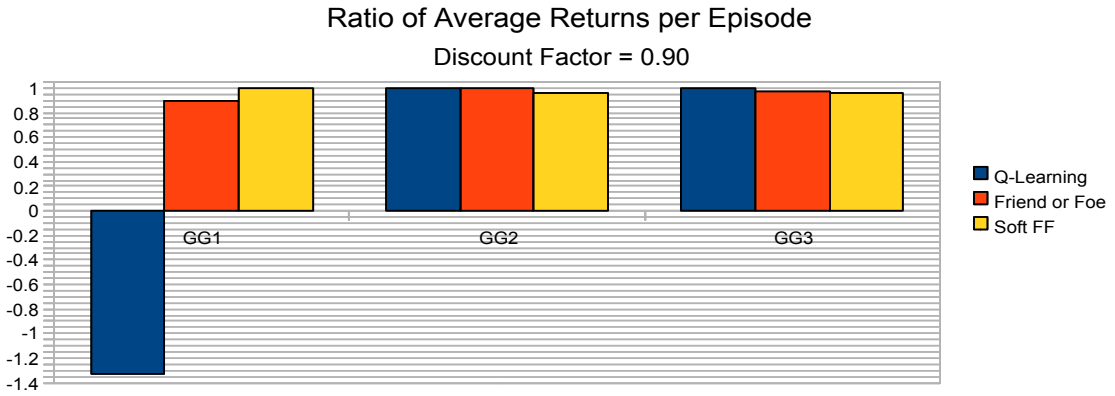

Figure 5.2: Average Return (Discount Factor = 0.90)

of 100 only slightly more often than the Q-Learning algorithm. The optimal solution is found when both agents receive the maximum reward of 100 in the fewest steps, which is 4 steps. The unmodified Friend or Foe algorithm resulted in the magority of the episodes ending with only one agent reaching the goal.

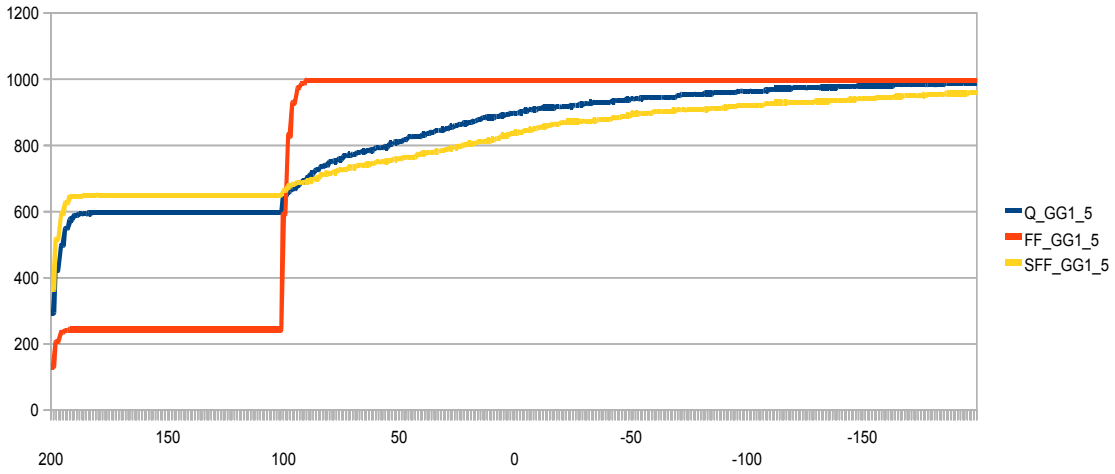

Figure 5.3: Number of Episodes with Reward $\geq x$ (GG1 Discount Factor = 0.50)

In grid game 2 (Fig 5.1), the Soft Friend or Foe algorithm was less likely than the other two algorithms to find the optimal solution for the pair of agents and more likely to find the optimal solution for a single agent (Figure E.2). The optimal solution for the pair is acheived when both agents enter the goal from the side and get a reward of 120 each in 3 steps. The optimal solution for a single agent results when one agent

enters the goal from below and receives a reward of 125 while the other agent enters from the side and receives a reward of 100 in 3 steps.

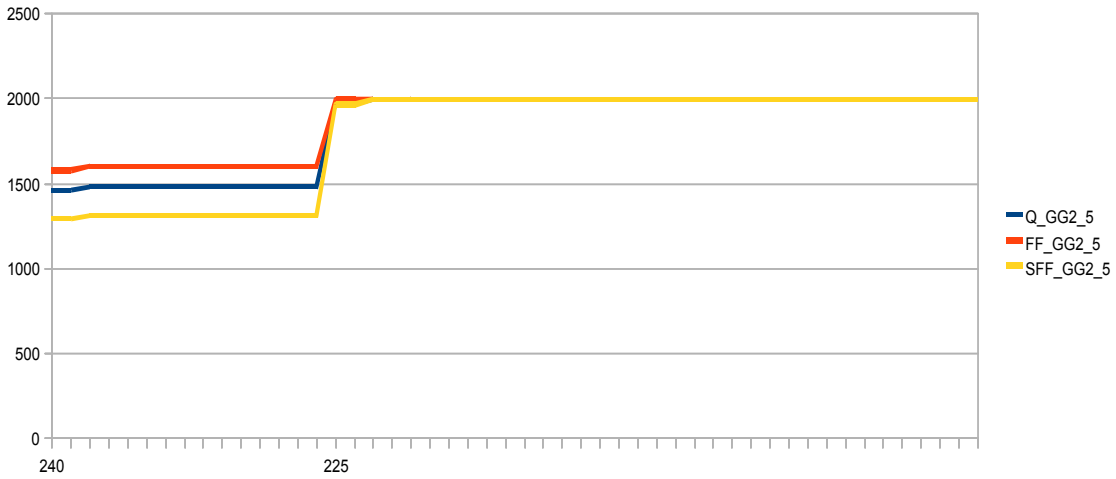

Figure 5.4: Number of Episodes with Reward $\geq x$ (GG2 Discount Factor = 0.50)

In grid game 3 (Fig 5.1), the algorithms gave similar results. The Soft Friend or Foe algorithm found the optimal solution only slightly more frequently than the other two algorithms (Figure E.3). The optimal solution for the pair is achieved when both agents enter the goal in 3 steps receiving 100 points each and 200 for the episode. More often only one agent reached the goal for a return of 100.

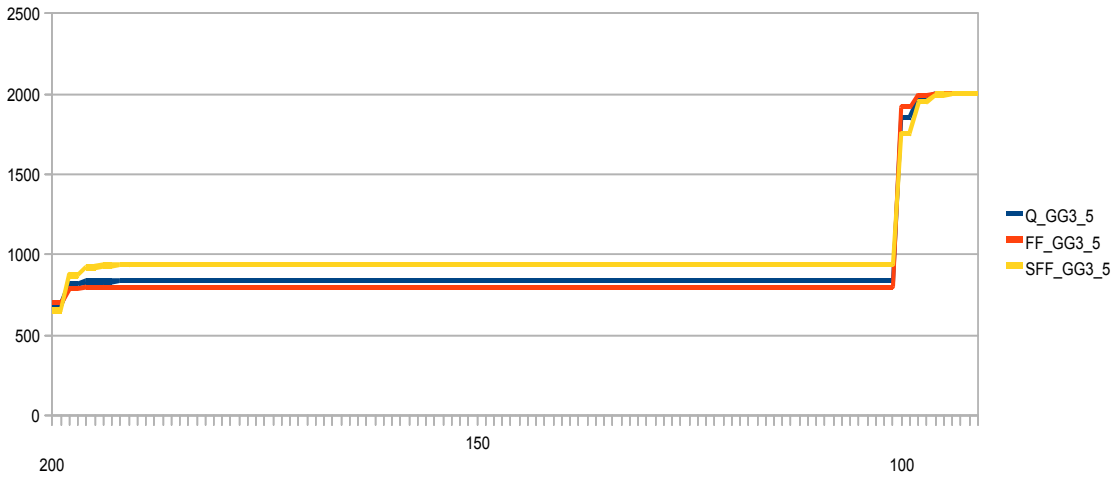

Figure 5.5: Number of Episodes with Reward $\geq x$ (GG3 Discount Factor = 0.50)

When the discount factor was increase to 0.90 to allow for more acceptable long term rewards in grid game 1, the Q-Learning policy resulted in disastrous results, but the Soft Friend or Foe algorithm maintained good results.

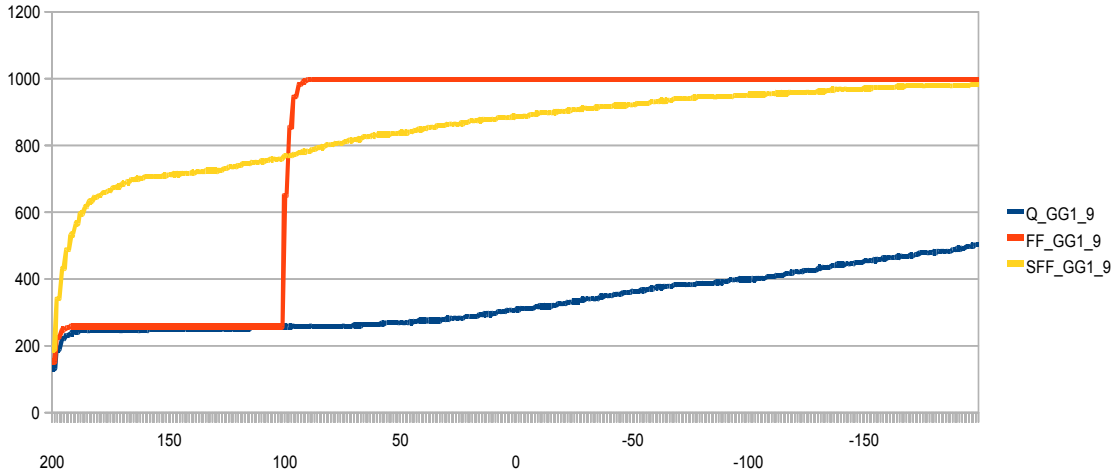

Figure 5.6: Number of Episodes with Reward $\geq x$ (GG1 Discount Factor = 0.90)

5.2.2 Policies

Appendix L summarizes the state action value estimates. From those estimates the following policies for the initial states shown in figure 5.1 are derived. The policies shown here are the policies that result in the returns that are discussed in the previous paragraphs. The equation used to derive the policy from state action value estimates is the softmax equation [6, pg. 30].

In grid game 1 Soft Friend or Foe and Q-Learning result in almost identical policies at the initial state. Friend or Foe is so pessimistic that all of the options from the initial state look just as bad and so Friend or Foe results in more collisions from the initial state.

Soft Friend or Foe results in the most aggressive policy for grid game 2. Friend or Foe takes the safe route since agents are defined as Foes. The results show that agents following a policy that results from the Friend or Foe algorithm take the middle path

Table 5.1: Grid Game 1 Policy for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action			
		UP		LEFT	
		Alg.	Probability	Alg.	Probability
Agent 1 Action	UP	Q	0.53	Q	0.20
		SFF	0.51	SFF	0.20
		FF	0.28	FF	0.25
	RIGHT	Q	0.20	Q	0.07
		SFF	0.21	SFF	0.08
		FF	0.25	FF	0.22

less than one percent of the time. The policies must become more aggressive when the agents get closer to the goal.

If the discount factor is increased agents do become more aggressive, and the average return suffers. Collisions increase and agents more often attempt to increase their own score which results in a lower combined score.

In grid game 3 Soft Friend or Foe is the most aggressive. The results do not vary significantly among the three algorithms.

5.2.3 Convergence

Soft Friend or For seems to converge more quickly than the other two algorithms (Figures: 5.7,5.8,5.9).

5.2.4 Speed

Q-Learning was the least computationally expensive, with Friend or Foe and Soft Friend or Foe nearly the same in the test cases (Figure 5.10).

Table 5.2: Grid Game 2 Policy for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action			
		UP		LEFT	
		Alg.	Probability	Alg.	Probability
Agent 1 Action	UP	Q	0.88	Q	0.06
		SFF	0.79	SFF	0.10
		FF	0.98	FF	0.01
	RIGHT	Q	0.06	Q	0.00
		SFF	0.10	SFF	0.01
		FF	0.01	FF	0.00

The Soft Friend or Foe algorithm was expected to be more computationally expensive because the correlation must be updated for every pair of agents every iteration. The small number of states and agents was probably not enough to demonstrate any speed difference between Soft Friend or Foe and the original Friend or Foe.

5.2.5 Memory Usage

There was not a significant difference in memory usage among the three algorithms compared. Memory requirements were expected to be greater for the Soft Friend or Foe algorithm because the correlation must be stored for all agent pairs for every state. My advisor and I suspect that the memory requirements of the program used to evaluate the algorithms far exceeded the memory requirements of each of the algorithms because of the simplicity of the environments used.

Table 5.3: Grid Game 3 Policy for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action			
		UP		LEFT	
		Alg.	Probability	Alg.	Probability
Agent 1 Action	UP	Q	0.46	Q	0.22
		SFF	0.32	SFF	0.25
		FF	0.55	FF	0.15
	RIGHT	Q	0.21	Q	0.11
		SFF	0.25	SFF	0.18
		FF	0.23	FF	0.07

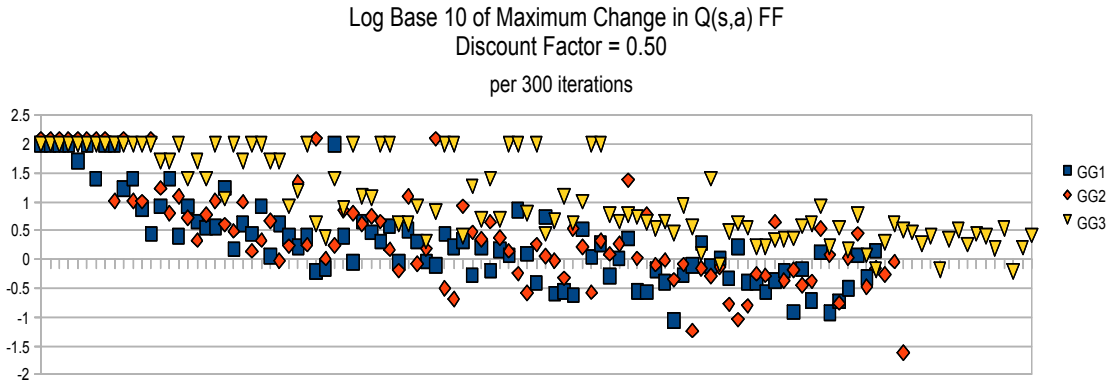

Figure 5.7: Convergence of State Action Value Estimates for Friend or Foe Algorithm

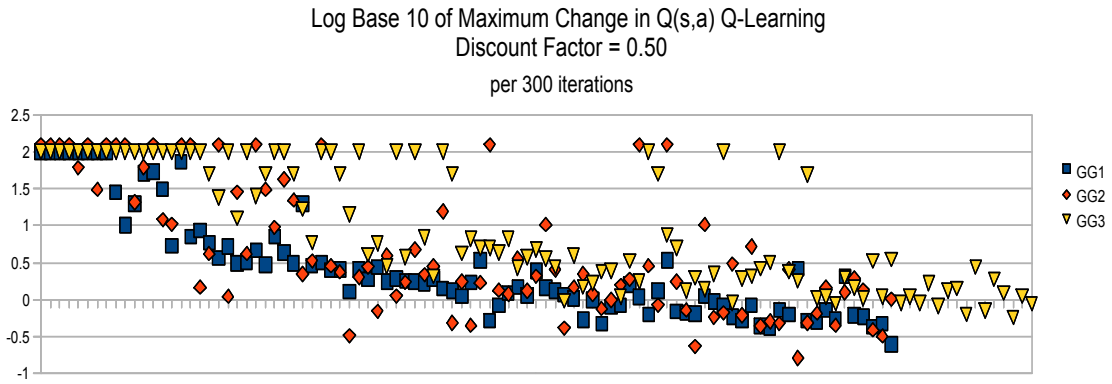

Figure 5.8: Convergence of State Action Value Estimates for Q-Learning Algorithm

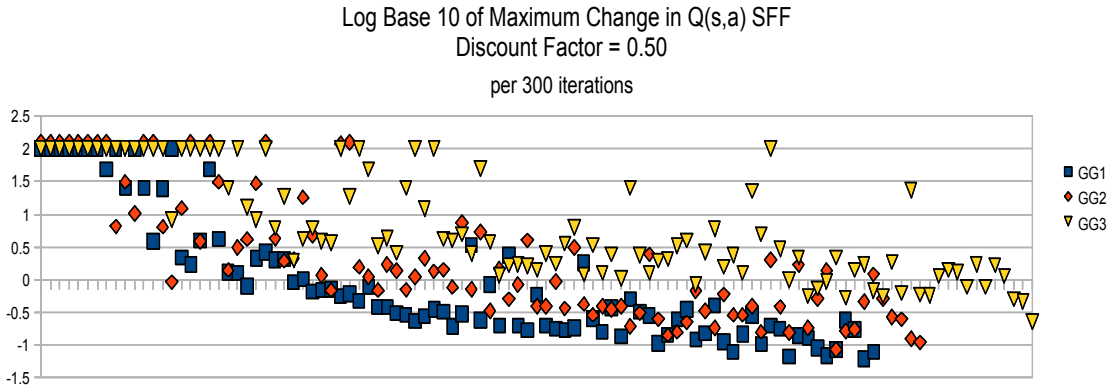

Figure 5.9: Convergence of State Action Value Estimates for Soft Friend or Foe Algorithm

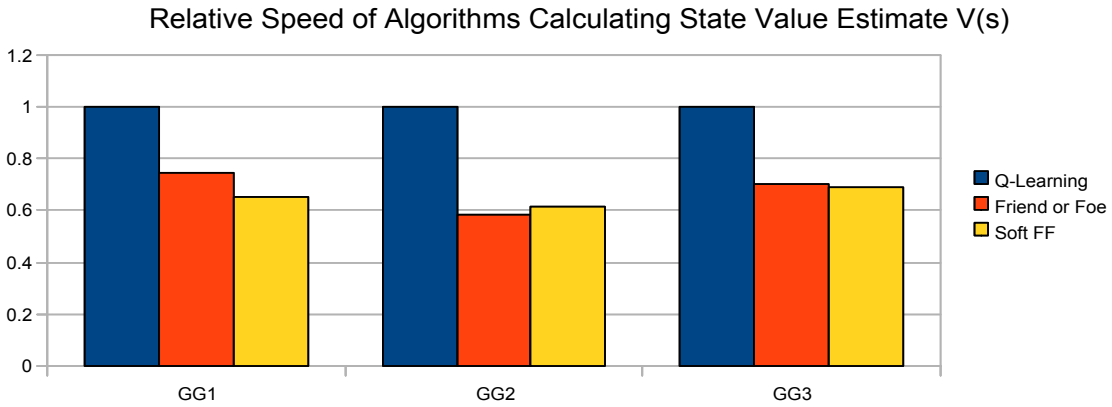

Figure 5.10: Relative Speed in Determining State Value Estimate V(s)

CHAPTER 6

SUMMARY

The Soft Friend or Foe was shown to converge faster than the other two algorithms and get returns equal to or greater than returns received using Q-Learning. Soft Friend or Foe received returns as good as Friend or Foe in all environments.

For future work, I think that the Soft Friend or Foe algorithm can be improved. Also it would be interesting to see how the memory usage and computation speed scale with more complex environments. For completeness, it would be nice to compare Soft Friend or Foe with the Nash-Q learning that Hu and Wellman presented[3].

In this experiment, correlation was calculated on a per state basis using immediate returns. I think that causes the big picture to be lost since only immediate returns are considered when determining the correlation. In these environments the only positive rewards are received when the final state is reached. Cases where both states are close to their goal at the same time result in a high correlation of returns. When following the paths of greatest expected rewards, in these environments, the agents often end up in that scenario. Also all negative and most zero rewards are strongly correlated.

Better results might be achieved with the Soft Friend or Foe algorithm if the estimated value of the next state, $V(x, s)$, is factored into the correlation. Another option is calculating correlation for the entire environment; not on a state by state basis. I predict that the first option of incorporating state value estimates would give the best results because it allows the agent to identify states where the two agents are opposed in the long term. The second option of using correlation for the entire environment would use less memory and might give better results than the method

used in this experiment.

Another possible improvement would be to consider the combined returns when agents are highly correlated. Currently if agents are highly correlated, then higher weights are assigned to actions that benefit the primary agent more. It might be better to grant higher weights to actions that result in a higher combined score for the two agents.

Finally the weighting can be improved. When considering the relative goodness of an action, the point half way between the lower and upper bounds of the range of estimated values is used as the fulcrum between a good and bad return. It would be better to use the median or mean value.

Rather than using a linear function to weight the estimated values that result from varying an action, it might be better to use an S shaped curve so that weights stay very low until the fulcrum point is reached. Weights would then quickly increase to 0.5 at the fulcrum and quickly increase to a value close to 1 where they would stay for the remainder of the range. It might be better to move the transition past the fulcrum to filter out even more choices.

BIBLIOGRAPHY

[1] M. L. Littman, "Friend-or-foe Q-learning in general-sum games," in *Proc. 18th International Conf. on Machine Learning*, pp. 322–328, Morgan Kaufmann, San Francisco, CA, 2001.

[2] J. Hu and M. P. Wellman, "Multiagent reinforcement learning: theoretical framework and an algorithm," in *Proc. 15th International Conf. on Machine Learning*, pp. 242–250, Morgan Kaufmann, San Francisco, CA, 1998.

[3] J. Hu and M. P. Wellman, "Nash q-learning for general-sum stochastic games," *J. Mach. Learn. Res.*, vol. 4, pp. 1039–1069, 2003.

[4] R. McKelvey and A. McLennan, "Computation of equilibria in finite games," 1996.

[5] J. Nash, "Non-cooperative games," *The Annals of Mathematics*, vol. 54, pp. 286–295, sep 1951.

[6] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[7] C. Szepesvari and M. Littman, "A unified analysis of value-function-based reinforcement-learning algorithms," 1997.

[8] B. von Stengel, "Computing equilibria for two-person games," 1996.

[9] A. Greenwald and K. Hall, "Correlated-q learning," in *Proceedings of the Twentieth International Conference on Machine Learning*, pp. 242–249, 2003.

[10] L. Lamport, *LaTeXUser's Guide and Reference Manual*. Addison Wesley, 2nd ed., 1994.

[11] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative learning," in *Readings in Agents* (M. N. Huhns and M. P. Singh, eds.), pp. 487–494, San Francisco, CA, USA: Morgan Kaufmann, 1997.

APPENDIX A

TEST SPECIFICATIONS

This data was collected with the following setup unless otherwise stated

Table A.1: Configuration

Environment	Algorithm	Episodes	Discount Factor	Iteration Limit	
				Offline	Evaluation
Grid Game 1	Q-Learning	1000	0.50	50	250
	SFF	1000	0.50	50	250
	FF	1000	0.50	50	250
Grid Game 2	Q-Learning	2000	0.50	50	250
	SFF	2000	0.50	50	250
	FF	2000	0.50	50	250
Grid Game 3	Q-Learning	2000	0.50	50	250
	SFF	2000	0.50	50	250
	FF	2000	0.50	50	250

APPENDIX B

AVERAGE RETURN PER EPISODE GRAPHS

The average return per episode is calculated by averaging the return for each episode that was run during the evaluation period, after the learning stage. The return for each episode is calculated by summing the returns for all iterations from the initial state to the final state. The return for an iteration is calculated by summing the returns for all agents from that iteration. In other words the return for an episode is the sum of all returns from start to finish of the episode. The results below are the averages over 1000 episodes for grid game 1 and 2000 episodes for grid games 2 and 3.

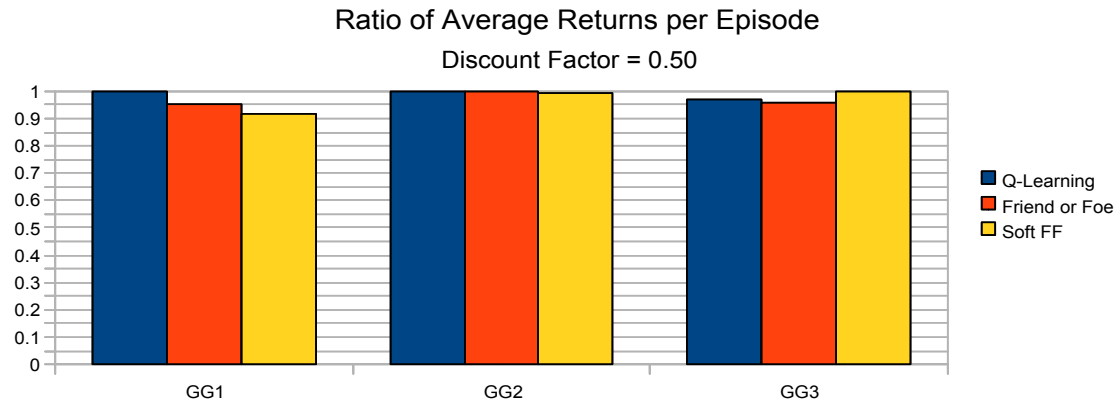

Figure B.1: Average Return Per Episode (Discount Factor = 0.50)

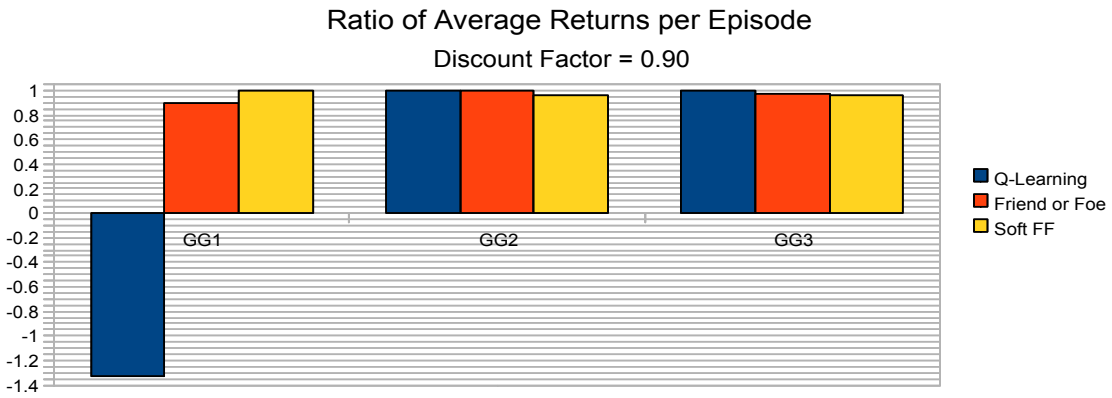

Figure B.2: Average Return Per Episode (Discount Factor = 0.90)

APPENDIX C

RATIO OF RETURNS IN COMPETITION

The following results were gathered by letting one agent use the Soft Friend or Foe algorithm to calculate the value of a state, and the other agent use either Q-Learning or Littman's Friend or Foe.

The average return per episode per agent is calculated by averaging the return per agent for each episode that was run during the evaluation period, after the learning stage. The return per agent for each episode is calculated by summing the returns for all iterations from the initial state to the final state for each agent. In other words the return per agent for an episode is the sum of all returns to a single agent from start to finish of the episode. The results below are ratios of the averages over 1000 episodes for grid game 1 and 2000 episodes for grid games 2 and 3.

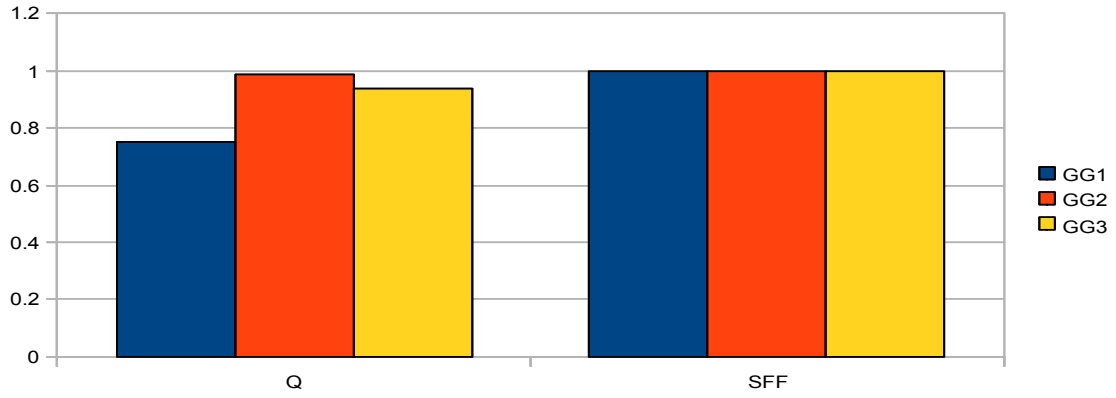

Figure C.1: Ratios of Average Return per Agent per Episode Q vs SFF (Discount Factor = 0.50)

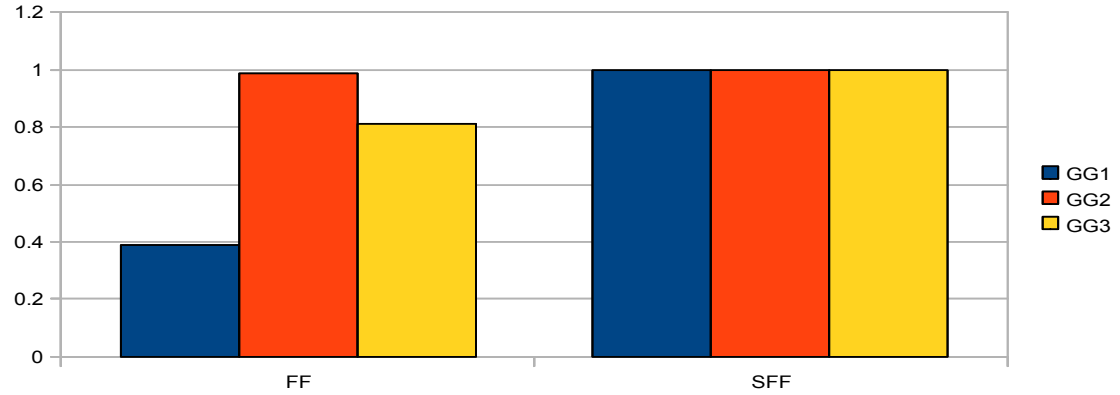

Figure C.2: Ratios of Average Return per Agent per Episode FF vs SFF (Discount Factor = 0.50)

APPENDIX D

RETURN HISTOGRAM GRAPHS

An average return per agent per episode histogram graph displays the number of episodes where an agent received a certain average return indicated on the x axis. The average return for an agent for an episode is calculated by summing all of the returns for an agent over a single episode and deviding that sum by the number of iterations in the episode.

An average return per episode histogram graph displays the number of episodes where the episode received a certain average return indicated on the x axis. The average return for an episode is calculated by summing all of the returns for all agents over a single episode and deviding that sum by the number of iterations in the episode.

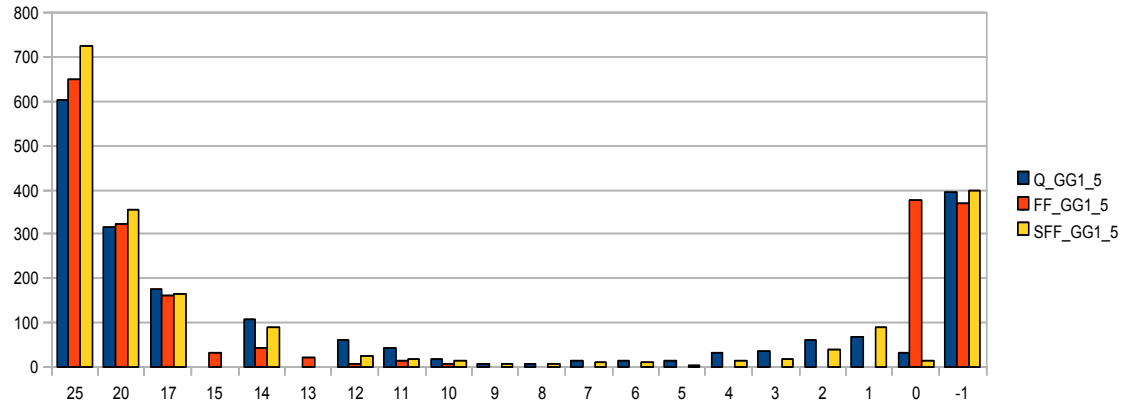

Figure D.1: Average Return Per Agent Per Episode Histogram (GG1 Discount Factor = 0.50)

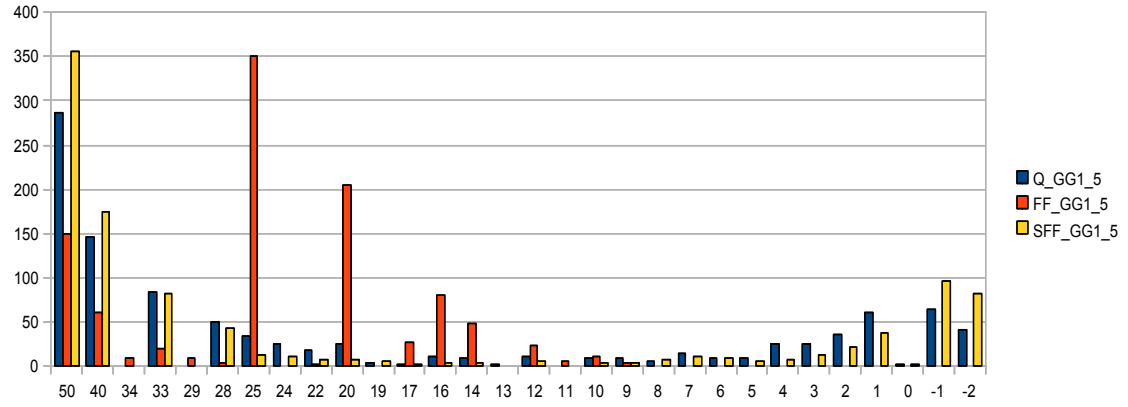

Figure D.2: Average Return Per Episode Histogram (GG1 Discount Factor = 0.50)

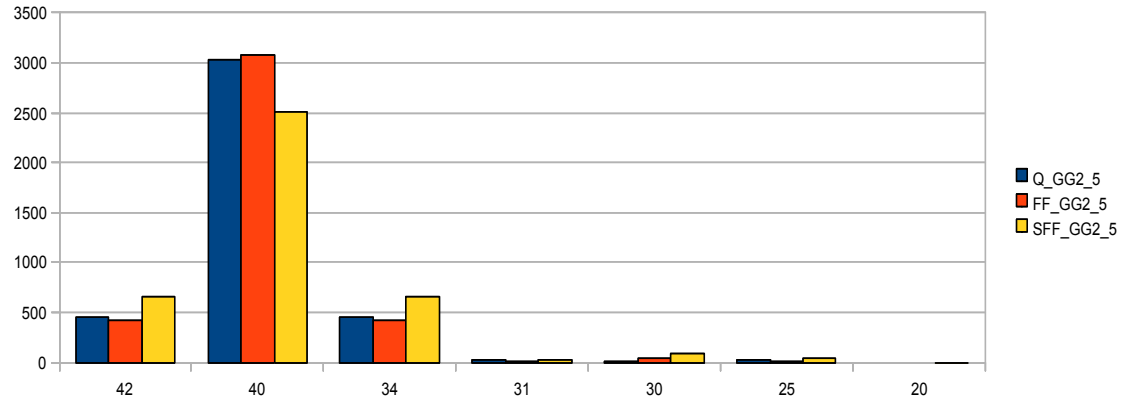

Figure D.3: Average Return Per Agent Per Episode Histogram (GG2 Discount Factor = 0.50)

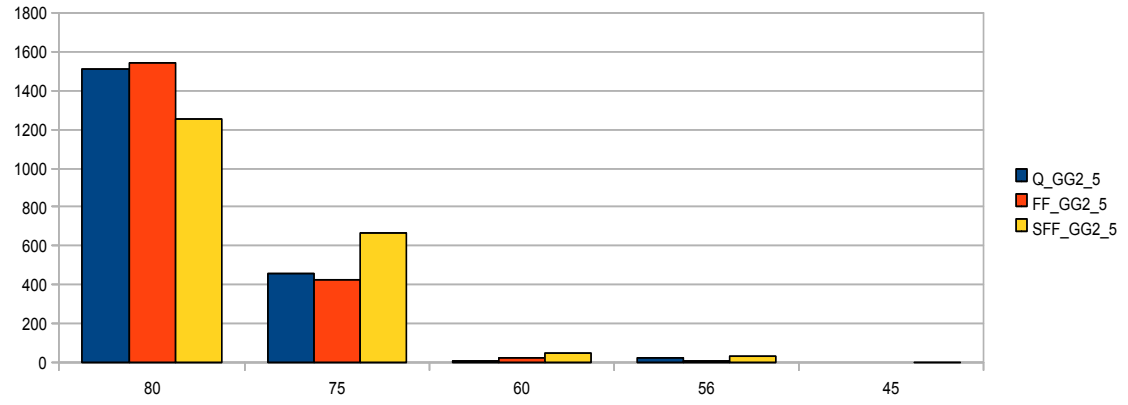

Figure D.4: Average Return Per Episode Histogram (GG2 Discount Factor = 0.50)

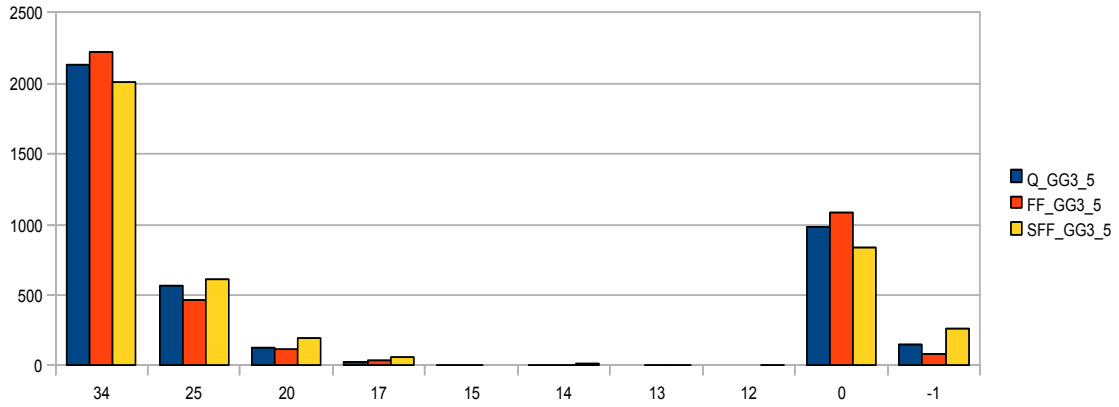

Figure D.5: Average Return Per Agent Per Episode Histogram (GG3 Discount Factor = 0.50)

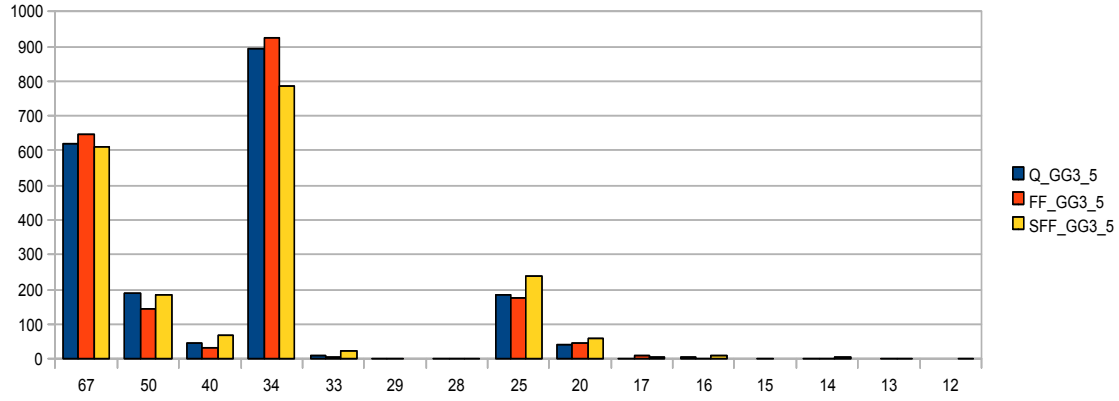

Figure D.6: Average Return Per Episode Histogram (GG3 Discount Factor = 0.50)

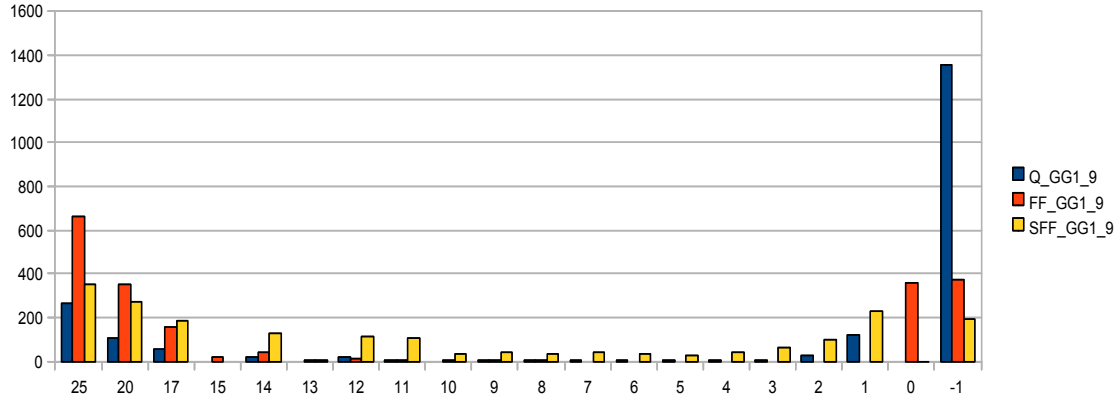

Figure D.7: Average Return Per Agent Per Episode Histogram (GG1 Discount Factor = 0.90)

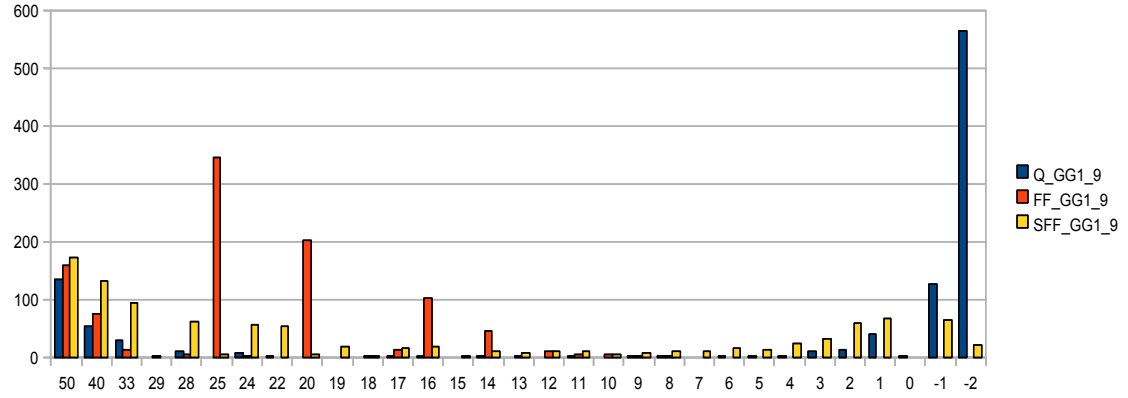

Figure D.8: Average Return Per Episode Histogram (GG1 Discount Factor = 0.90)

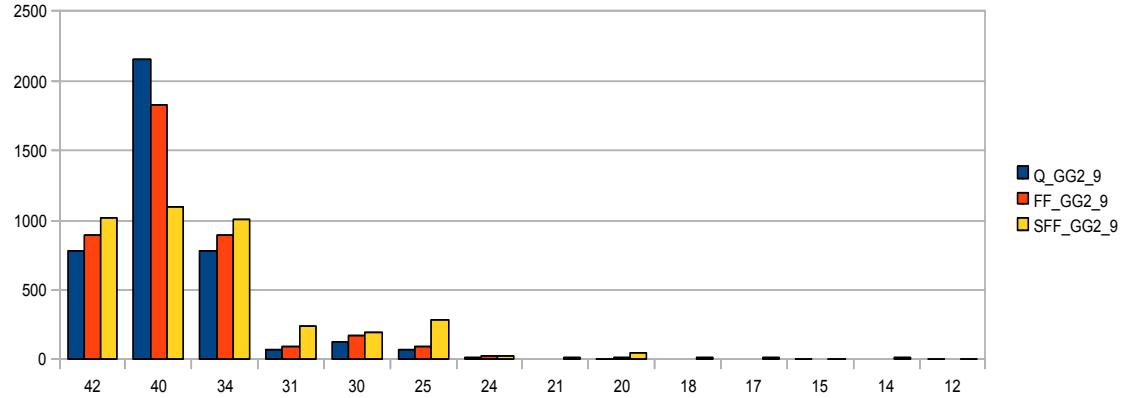

Figure D.9: Average Return Per Agent Per Episode Histogram (GG2 Discount Factor = 0.90)

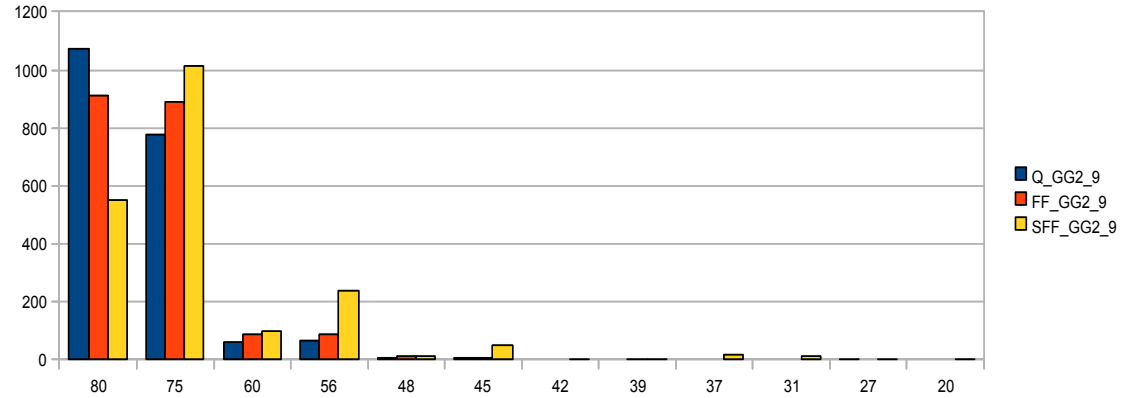

Figure D.10: Average Return Per Episode Histogram (GG2 Discount Factor = 0.90)

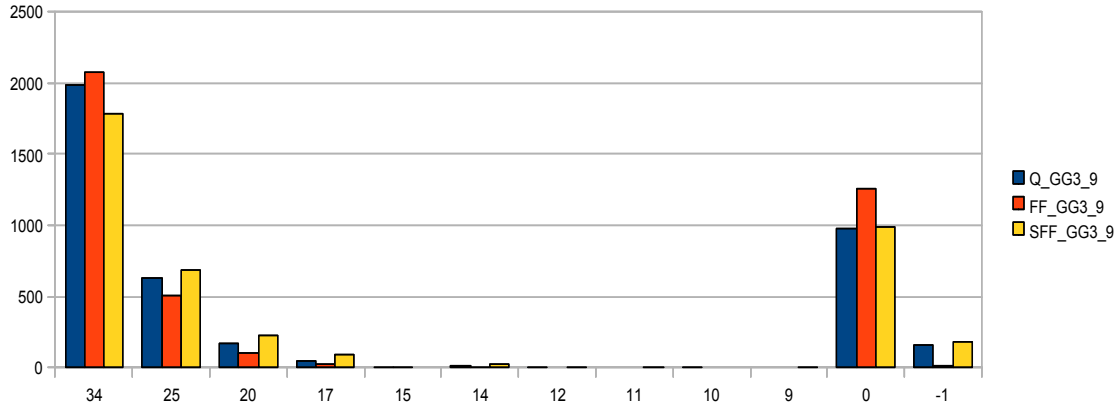

Figure D.11: Average Return Per Agent Per Episode Histogram (GG3 Discount Factor = 0.90)

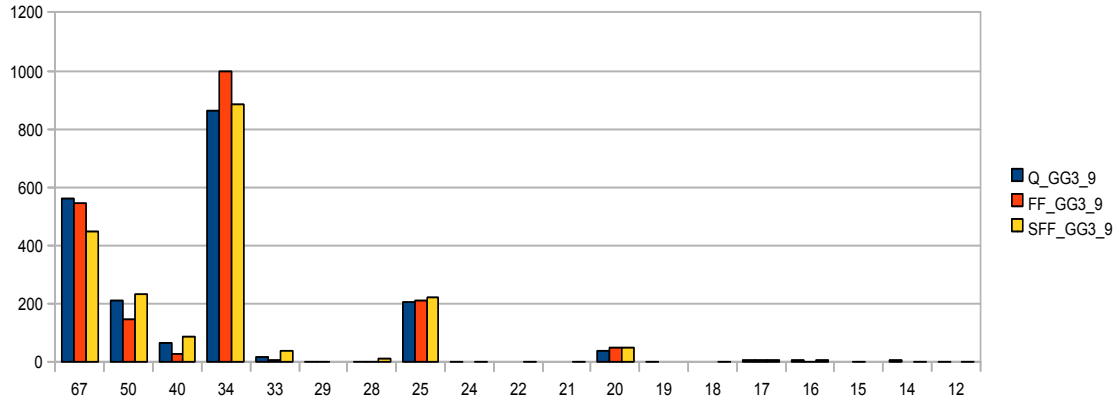

Figure D.12: Average Return Per Episode Histogram (GG3 Discount Factor = 0.90)

APPENDIX E

RETURN AGGREGATE HISTOGRAM GRAPHS

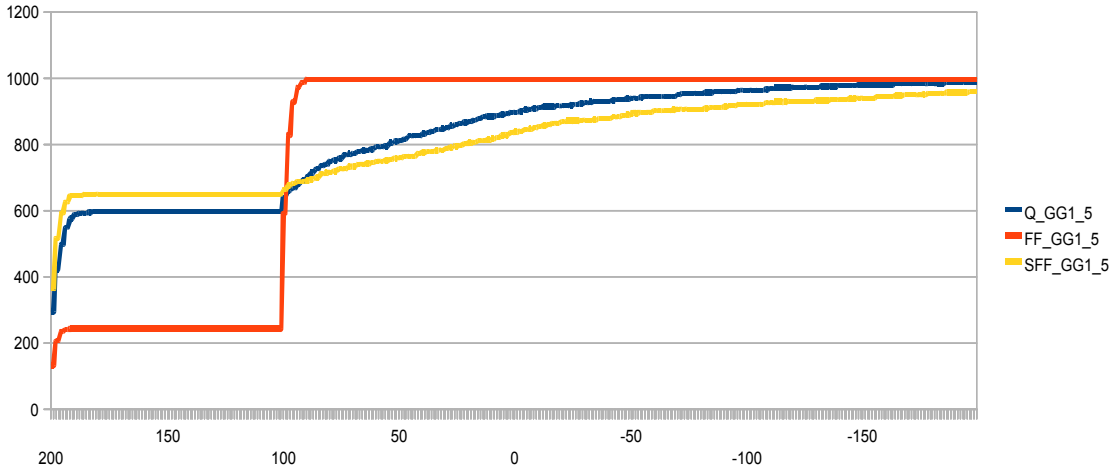

Figure E.1: Number of Episodes with Reward $\geq x$ (GG1 Discount Factor = 0.50)

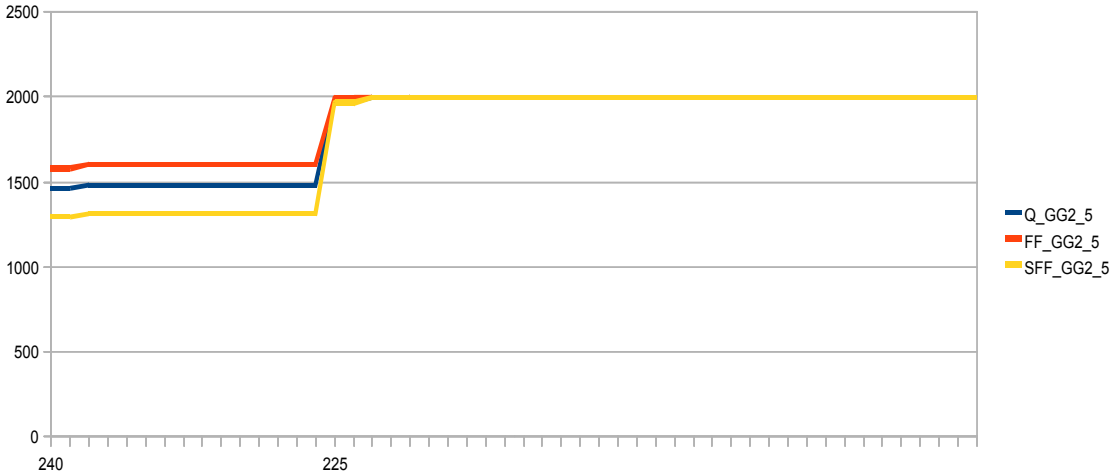

Figure E.2: Number of Episodes with Reward $\geq x$ (GG2 Discount Factor = 0.50)

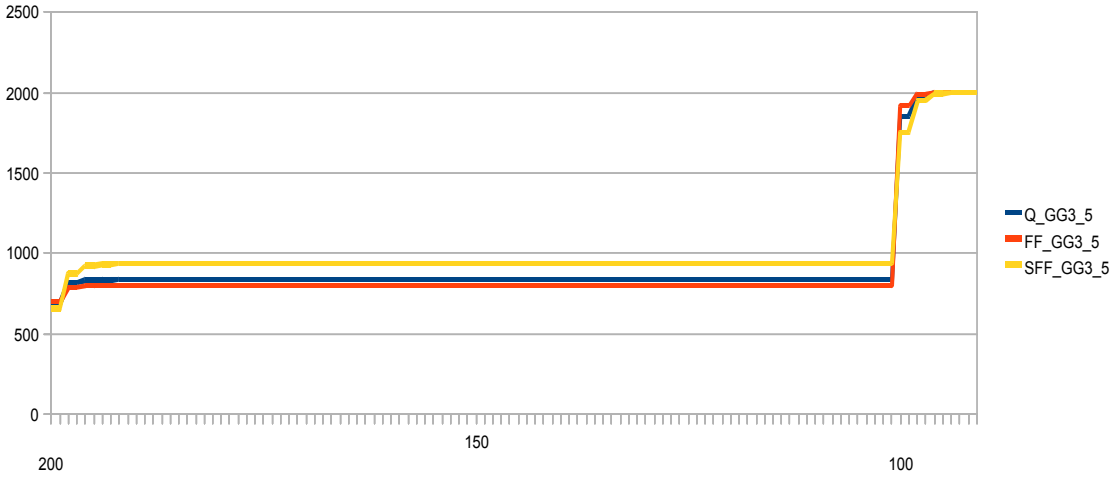

Figure E.3: Number of Episodes with Reward $\geq x$ (GG3 Discount Factor = 0.50)

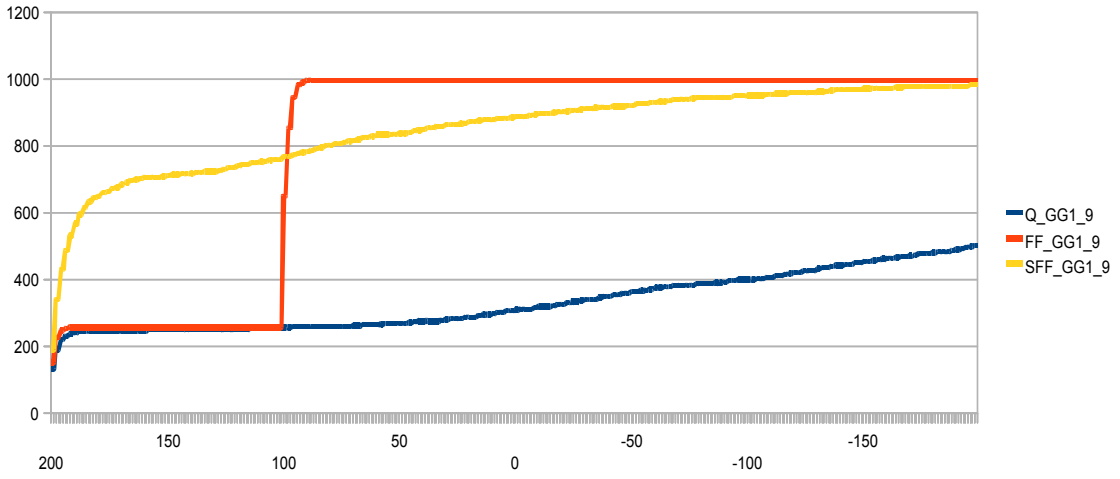

Figure E.4: Number of Episodes with Reward $\geq x$ (GG1 Discount Factor = 0.90)

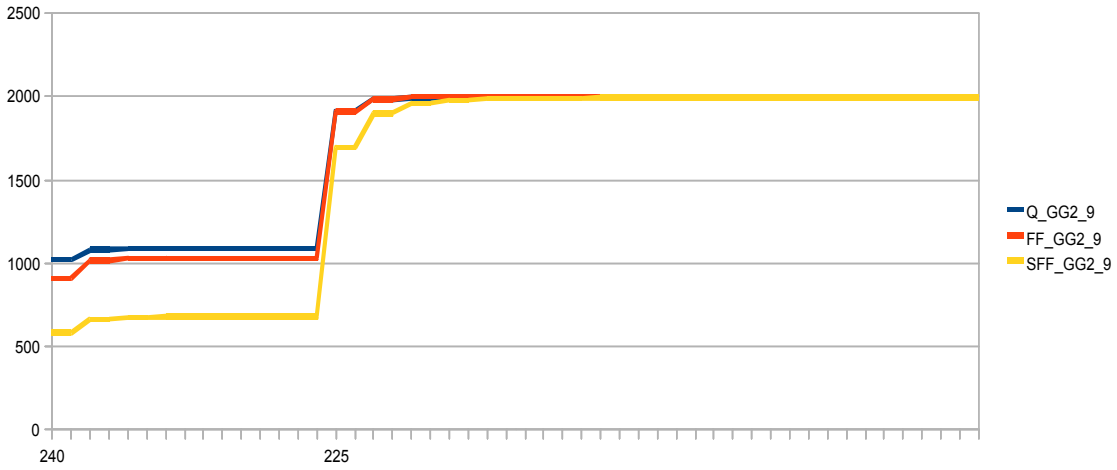

Figure E.5: Number of Episodes with Reward $\geq x$ (GG2 Discount Factor = 0.90)

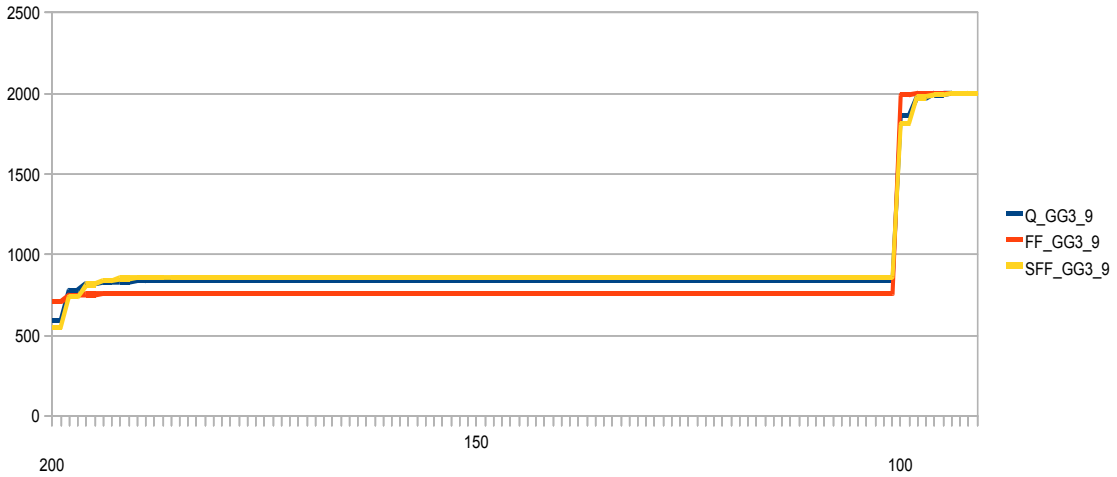

Figure E.6: Number of Episodes with Reward $\geq x$ (GG3 Discount Factor = 0.90)

APPENDIX F

COMPUTATIONAL SPEED GRAPHS

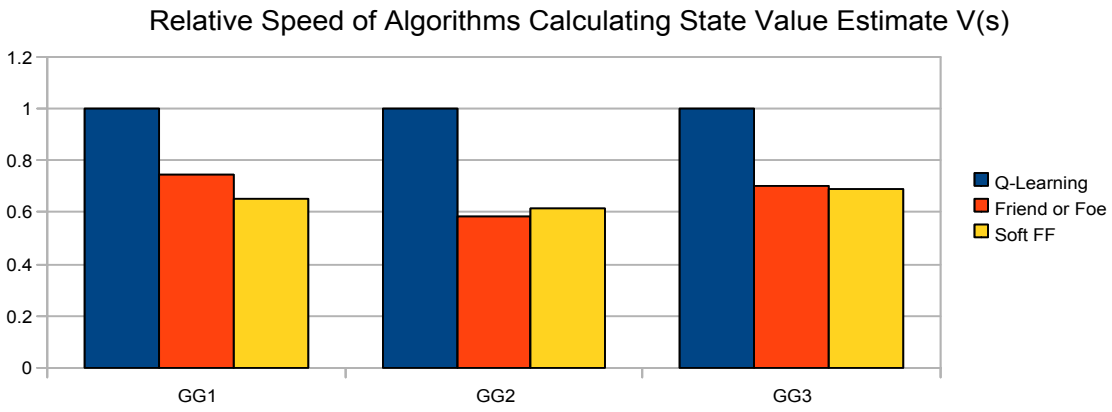

Figure F.1: Relative Speed in Determining State Value Estimate V(s)

APPENDIX G

CONVERGENCE GRAPHS DISCOUNT FACTOR 0.50

Convergence graphs show the maximum change in a state action sequence estimate every 300 iterations. The x axis values are indexes into every 300th iteration. The y value is the greatest change in a state action sequence estimate over a 300 iteration period.

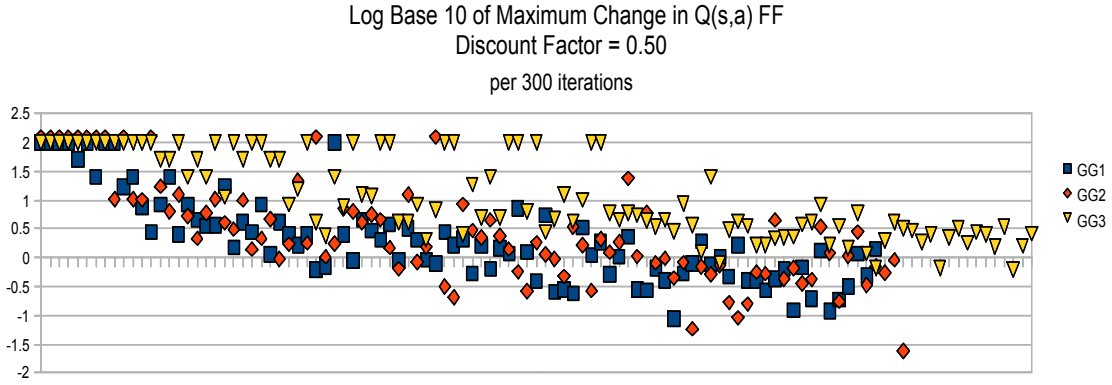

Figure G.1: Convergence of State Action Value Estimates for Friend or Foe Algorithm

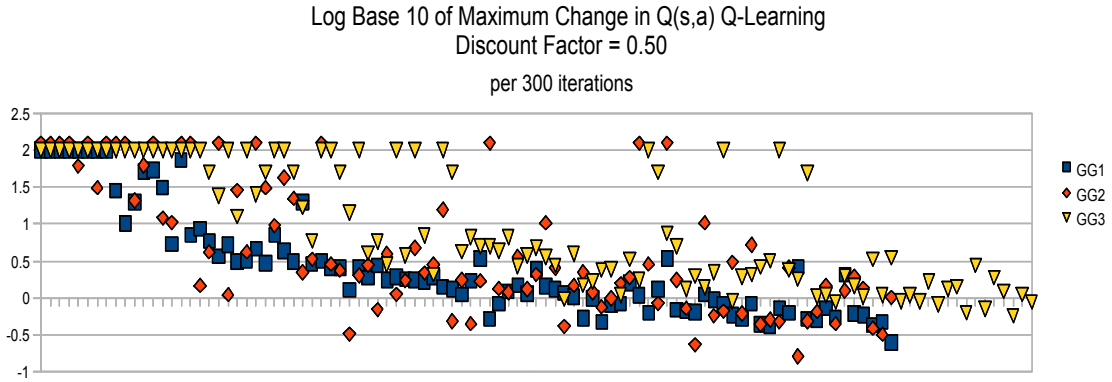

Figure G.2: Convergence of State Action Value Estimates for Q-Learning Algorithm

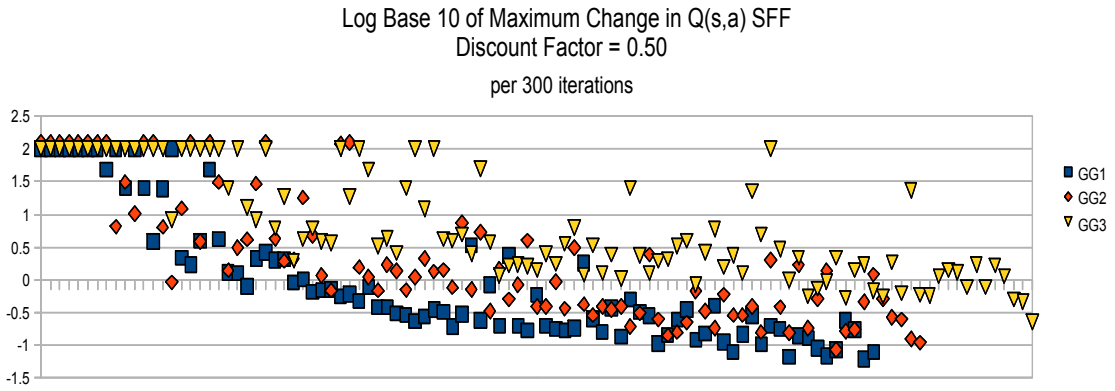

Figure G.3: Convergence of State Action Value Estimates for Soft Friend or Foe Algorithm

APPENDIX H

CONVERGENCE GRAPHS DISCOUNT FACTOR 0.90

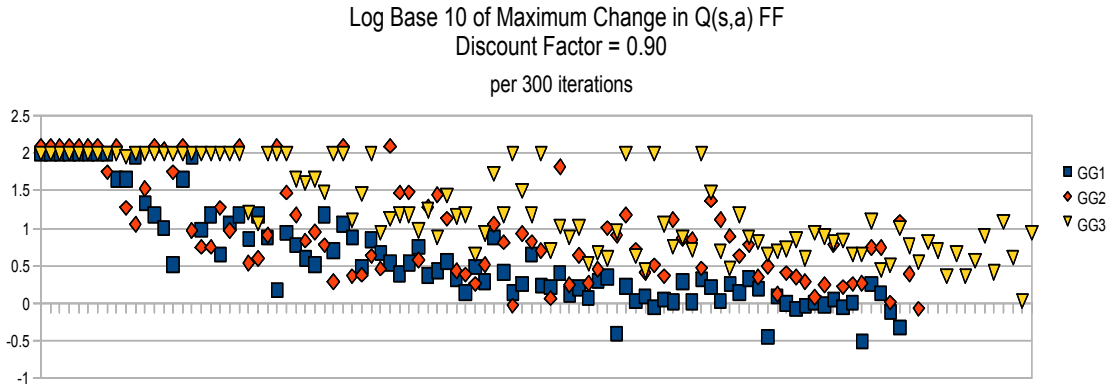

Figure H.1: Convergence of State Action Value Estimates for Friend or Foe Algorithm

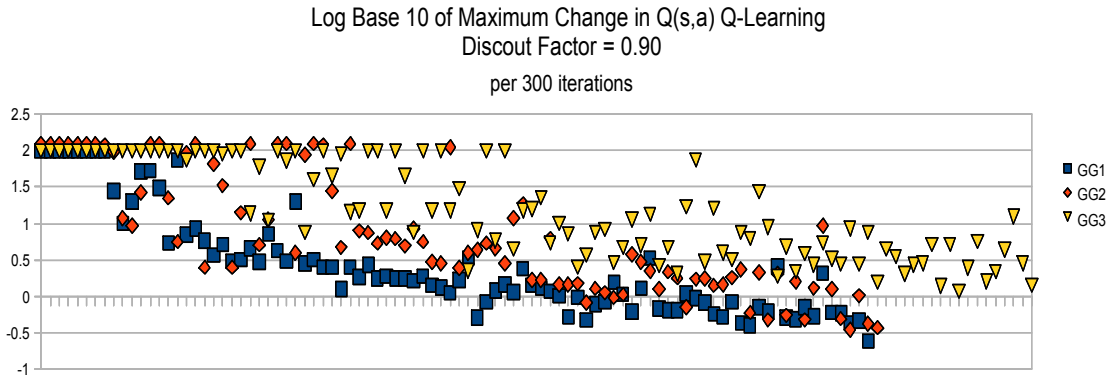

Figure H.2: Convergence of State Action Value Estimates for Q-Learning Algorithm

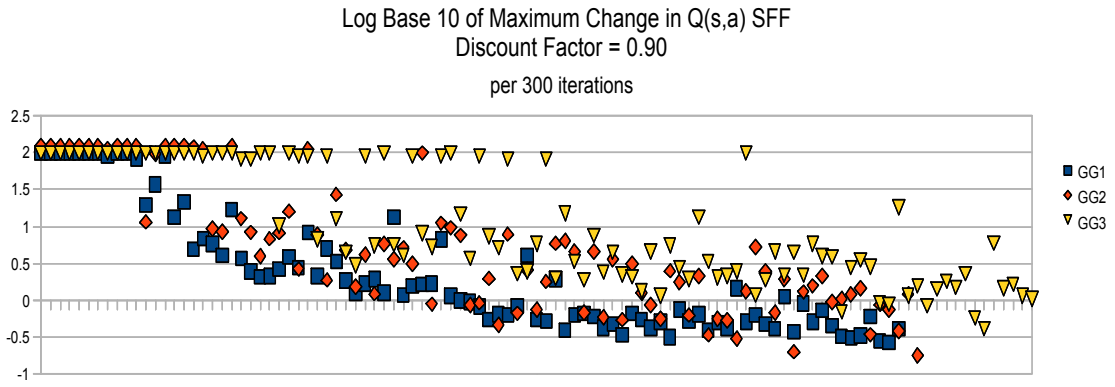

Figure H.3: Convergence of State Action Value Estimates for Soft Friend or Foe Algorithm

APPENDIX I

AVG REWARDS

Table I.1: Average Rewards Per Episode in Simple Grid Games (Discount Factor = 0.50)

Algorithm	Grid Game 1		Grid Game 2		Grid Game 3	
	Agent 1	Agent 2	Agent 1	Agent 2	Agent 1	Agent 2
Q-Learning	66.44	62.64	118.12	117.97	69.93	71.63
Soft Friend or Foe	57.60	60.90	117.58	117.18	71.76	74.61
Friend or Foe	61.66	61.06	118.67	118.32	71.50	68.50

Table I.2: Standard Deviation of Rewards Per Episode in Simple Grid Games (Discount Factor = 0.50)

Algorithm	Grid Game 1		Grid Game 2		Grid Game 3	
	Agent 1	Agent 2	Agent 1	Agent 2	Agent 1	Agent 2
Q-Learning	58.73	61.35	7.09	7.22	45.76	44.99
Soft Friend or Foe	73.88	71.67	8.40	8.28	44.83	43.30
Friend or Foe	48.48	48.62	6.17	6.49	45.07	46.39

Table I.3: Average Rewards Per Episode in Simple Grid Games (Discount Factor = 0.90)

Algorithm	Grid Game 1		Grid Game 2		Grid Game 3	
	Agent 1	Agent 2	Agent 1	Agent 2	Agent 1	Agent 2
Q-Learning	-94.11	-89.61	116.15	116.83	70.55	70.80
Soft Friend or Foe	66.72	71.52	113.56	115.71	70.63	71.68
Friend or Foe	62.48	61.98	116.89	115.61	70.46	67.31

Table I.4: Standard Deviation of Rewards Per Episode in Simple Grid Games (Discount Factor = 0.90)

Algorithm	Grid Game 1		Grid Game 2		Grid Game 3	
	Agent 1	Agent 2	Agent 1	Agent 2	Agent 1	Agent 2
Q-Learning	141.40	142.28	9.39	9.09	45.42	45.28
Soft Friend or Foe	59.81	52.18	11.89	11.08	45.25	44.79
Friend or Foe	48.24	48.44	9.20	9.70	45.59	46.88

APPENDIX J

AVG REWARDS PER AGENT WITH MIXED ALGORITHMS

Table J.1: Q-Learning vs. Soft Friend or Foe Average Rewards Per Agent Per Episode in Simple Grid Games (Discount Factor = 0.50)

Algorithm	Grid Game 1	Grid Game 2	Grid Game 3
Q-Learning	50.75	117.25	68.47
Soft Friend or Foe	67.45	118.65	72.97

Table J.2: Q-Learning vs. Soft Friend or Foe Standard Deviation of Rewards Per Episode Per Agent in Simple Grid Games (Discount Factor = 0.50)

Algorithm	Grid Game 1	Grid Game 2	Grid Game 3
Q-Learning	79.2	7.79	46.35
Soft Friend or Foe	64.36	6.72	44.27

Table J.3: Friend or Foe vs. Soft Friend or Foe Average Rewards Per Agent Per Episode in Simple Grid Games (Discount Factor = 0.50)

Algorithm	Grid Game 1	Grid Game 2	Grid Game 3
Friend or Foe	38.01	116.93	63.41
Soft Friend or Foe	98.21	118.71	78.01

Table J.4: Friend or Foe vs. Soft Friend or Foe Standard Deviation of Rewards Per Episode Per Agent in Simple Grid Games (Discount Factor = 0.50)

Algorithm	Grid Game 1	Grid Game 2	Grid Game 3
Friend or Foe	48.73	8.1	48.11
Soft Friend or Foe	10.29	6.8	41.26

APPENDIX K

RESOURCE UTILIZATION

* Memory usage was uniform over all algorithms and games.

Table K.1: Time (ms) to Calculate Estimated State Value V(s) (Discount Factor = 0.50)

Algorithm	Grid Game 1	Grid Game 2	Grid Game 3
Q-Learning	173.72	234.22	267.04
Soft Friend or Foe	168.86	290.66	273.91
Friend or Foe	171.76	245.77	248.24

Table K.2: Time (ms) to Calculate Estimated State Value V(s) (Discount Factor = 0.90)

Algorithm	Grid Game 1	Grid Game 2	Grid Game 3
Q-Learning	173.72	161.15	??
Soft Friend or Foe	273.74	287.70	272.85
Friend or Foe	222.80	248.37	250.78

APPENDIX L

STATE ACTION VALUE ANALYSIS DISCOUNT FACTOR 0.50

Table L.1: Grid Game 1 Estimated State Action Value Q(s,a) Ratios for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	0.85	0.83	Q	1.00	0.57
		SFF	1.00	0.98	SFF	0.99	0.92
		FF	1.00	1.00	FF	1.00	1.00
	RIGHT	Q	0.62	1.00	Q	0.24	0.22
		SFF	0.95	1.00	SFF	0.38	0.37
		FF	1.00	1.00	FF	0.00	0.00

Table L.2: Grid Game 1 Estimated State Action Value Q(s,a) for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	7.06	6.43	Q	8.34	4.41
		SFF	11.30	11.23	SFF	11.16	10.60
		FF	0.00	0.00	FF	0.00	0.00
	RIGHT	Q	5.20	7.79	Q	1.98	1.71
		SFF	10.78	11.51	SFF	4.32	4.30
		FF	0.00	0.00	FF	-1.00	-1.00

Table L.3: Grid Game 2 Estimated State Action Value Q(s,a) Ratios for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	1.00	1.00	Q	0.97	0.81
		SFF	0.99	1.00	SFF	1.00	0.97
		FF	0.96	1.00	FF	1.00	0.00
	RIGHT	Q	0.79	1.00	Q	0.41	0.42
		SFF	0.99	0.98	SFF	0.46	0.45
		FF	0.00	1.00	FF	0.39	0.37

Table L.4: Grid Game 2 Estimated State Action Value Q(s,a) for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	29.63	29.02	Q	28.64	23.60
		SFF	30.64	30.55	SFF	30.82	29.70
		FF	23.85	22.87	FF	24.97	0.00
	RIGHT	Q	23.33	29.23	Q	12.10	12.15
		SFF	30.64	30.05	SFF	14.06	13.67
		FF	0.00	22.87	FF	9.67	8.51

Table L.5: Grid Game 3 Estimated State Action Value Q(s,a) Ratio for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	0.78	0.75	Q	0.82	1.00
		SFF	0.72	0.77	SFF	0.77	1.00
		FF	1.00	1.00	FF	0.81	0.75
	RIGHT	Q	1.00	0.79	Q	0.29	0.25
		SFF	1.00	0.75	SFF	0.40	0.43
		FF	0.97	0.81	FF	0.14	0.22

Table L.6: Grid Game 3 Estimated State Action Value Q(s,a) for initial state shown in figure 5.1 (Discount Factor = 0.50)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	15.51	14.46	Q	16.41	19.24
		SFF	16.76	18.00	SFF	17.91	23.42
		FF	10.24	12.23	FF	8.33	9.21
	RIGHT	Q	20.00	15.18	Q	5.71	4.73
		SFF	23.15	17.64	SFF	9.17	10.07
		FF	9.97	9.96	FF	1.47	2.72

APPENDIX M

STATE ACTION VALUE ANALYSIS DISCOUNT FACTOR 0.90

The figures in this table relate to the grid games described in section 5.1.3; see Fig 5.1.

Table M.1: Grid Game 1 Estimated State Action Value Q(s,a) Ratio for initial state shown in figure 5.1 (Discount Factor = 0.90)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	0.85	0.92	Q	1.00	0.65
		SFF	0.96	0.99	SFF	1.00	1.00
		FF	1.00	1.00	FF	1.00	1.00
	RIGHT	Q	0.69	1.00	Q	0.63	0.67
		SFF	0.92	1.00	SFF	0.81	0.80
		FF	1.00	1.00	FF	0.00	0.00

Table M.2: Grid Game 1 Estimated State Action Value Q(s,a) for initial state shown in figure 5.1 (Discount Factor = 0.90)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	47.10	50.35	Q	55.70	35.80
		SFF	66.72	64.03	SFF	69.57	64.85
		FF	0.00	0.00	FF	0.00	0.00
	RIGHT	Q	38.56	54.99	Q	34.83	36.71
		SFF	64.21	64.80	SFF	56.17	51.73
		FF	0.00	0.00	FF	-1.00	-1.00

Table M.3: Grid Game 2 Estimated State Action Value Q(s,a) Ratios for initial state shown in figure 5.1 (Discount Factor = 0.90)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	0.93	0.96	Q	1.00	0.92
		SFF	1.00	1.00	SFF	0.99	0.98
		FF	0.99	1.00	FF	1.00	0.00
	RIGHT	Q	0.87	1.00	Q	0.73	0.79
		SFF	0.97	0.98	SFF	0.86	0.86
		FF	0.00	0.91	FF	0.67	0.61

Table M.4: Grid Game 2 Estimated State Action Value Q(s,a) for initial state shown in figure 5.1 (Discount Factor = 0.90)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	87.46	93.16	Q	94.30	89.56
		SFF	99.03	99.72	SFF	98.49	97.95
		FF	68.18	72.98	FF	68.56	0.00
	RIGHT	Q	82.33	97.08	Q	68.90	76.22
		SFF	95.93	97.76	SFF	85.16	85.53
		FF	0.00	66.43	FF	45.74	44.23

Table M.5: Grid Game 3 Estimated State Action Value Q(s,a) Ratios for initial state shown in figure 5.1 (Discount Factor = 0.90)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	0.81	0.80	Q	0.86	1.00
		SFF	0.95	0.94	SFF	0.92	1.00
		FF	1.00	1.00	FF	0.97	0.81
	RIGHT	Q	1.00	0.88	Q	0.60	0.59
		SFF	1.00	0.95	SFF	0.81	0.84
		FF	0.89	0.95	FF	0.63	0.51

Table M.6: Grid Game 3 Estimated State Action Value Q(s,a) for initial state shown in figure 5.1 (Discount Factor = 0.90)

		Agent 2 Action					
		UP			LEFT		
		Alg.	Agent 1 Reward	Agent 2 Reward	Alg.	Agent 1 Reward	Agent 2 Reward
Agent 1 Action	UP	Q	55.52	53.08	Q	58.59	66.10
		SFF	73.38	72.05	SFF	71.61	76.97
		FF	37.34	33.63	FF	36.15	27.31
	RIGHT	Q	68.13	58.44	Q	40.95	38.90
		SFF	77.62	73.33	SFF	62.85	64.29
		FF	33.41	31.88	FF	23.55	17.08

APPENDIX N

ALGORITHMS

N.1 Main

1. $A(x,s) \leftarrow Q(x,s)\ XOR\ FF(x,s)\ XOR\ SFF(x,s)$
2. $H = ()$
3. for each agent $x \in X$
   (a) $R(x) \leftarrow ()$
   (b) for each state $s \in S$
      i. $R(x,s) \leftarrow H(s) \leftarrow ()$
      ii. for each action $a \in \delta_A(x,s)$
         A. $R(x,s,a) \leftarrow H(x,s,a) \leftarrow ()$
      iii. for each action set $\Lambda(s) \in k(s)$
         A. $R(x,s,\Lambda(s)) \leftarrow H(s,\Lambda(s)) \leftarrow ()$, $Q(x,s,\Lambda(s)) \leftarrow 0$
4. $\gamma \leftarrow .95$
5. state $s \leftarrow s_1 | s_1 \in S$
6. $H \leftarrow Hs$
7. for t in 1..N where N is the number of iterations desired
   (a) $\Lambda(s) \leftarrow ()$
   (b) for each agent $x \in X$

i. agent x selects action $a \in \delta_A(x,s)$ according to $\pi(x,s)$

ii. $\Lambda(s) \leftarrow \Lambda(s)(a)$

(c) $s' \leftarrow f_p(s, \Lambda(s), \delta_p)$

(d) $H \leftarrow Hs'$

(e) $H(s) \leftarrow H(s)\, s'$

(f) $H(s, \Lambda(s)) \leftarrow H(s, \Lambda(s))\, s'$

(g) for each agent $x \in X$

i. $f \leftarrow \delta_r(x, \Lambda(s), s, s')$, $r \leftarrow f$

ii. $R(x) \leftarrow R(x)(r)$

iii. $R(x,s) \leftarrow R(x,s)(r)$

iv. $R(x,s,a(x,\Lambda(s))) \leftarrow R(x,s,a(x,\Lambda(s)))(r)$

v. $R(x,s,\Lambda(s)) \leftarrow R(x,s,\Lambda(s))(r)$

vi. $H(x,s,a(x,\Lambda(s))) \leftarrow H(x,s,a(x,\Lambda(s)))\, s'$

vii. $k \leftarrow |R(x,s,\Lambda(s))|$, $r_k \leftarrow r$, $\alpha(k) \leftarrow 1/k$

viii. $A(x,s')$

ix. Set $Q(x,s,\Lambda(s))$ with equation 3.6

(h) $s \leftarrow s'$

N.2 Friend or Foe - FF(x,s)

1. $U \leftarrow ()$

2. $V \leftarrow ()$

3. for each agent $x_i \in X$

(a) if x_i is x or x_i is a friend of x then

i. $U \leftarrow U(x_i)$

(b) else

i. $V \leftarrow V(x_i)$

4. $B \leftarrow X$

5. for each agent $v \in V$

(a) for all $w \in \psi(v, \Lambda(B, s))$

i. $w' \leftarrow \Phi(v, w)$

ii. $Q(x, s, w') \leftarrow min_{z \in \Psi(v,w)} Q(x, s, z)$

(b) $B \leftarrow B \bigcap \overline{v}$

6. for each agent $u \in U$

(a) for all $w \in \psi(u, \Lambda(B, s))$

i. $w' \leftarrow \Phi(u, w)$

ii. $Q(x, s, w') \leftarrow max_{z \in \Psi(u,w)} Q(x, s, z)$

(b) $B \leftarrow B \bigcap \overline{u}$

7. $V(x, s) \leftarrow Q(x, s, \{\})$

N.3 Soft Friend or Foe - SFF(x,s)

1. $B \leftarrow X$

2. while $|B| > 1$

(a) $y \leftarrow x_i | \min_{x_i \in B} \rho_k(x, x_i, s)$

(b) for all $w \in \psi(y, \Lambda(B, s))$

i. $w' \leftarrow \Phi(y, w)$

ii. $Q(x,s,w') \leftarrow \frac{\sum_{\Lambda(s)\in\Psi(y,w)} W(x,y,s,\Lambda(s))*Q(x,s,\Lambda(s))}{\sum_{\Lambda(s)\in\Psi(y,w)} W(x,y,s,\Lambda(s))}$

(c) $B \leftarrow B \cap \overline{y}$

3. $V(x,s) \leftarrow max_{w\in\Lambda(B,s)} Q(x,s,w)$

*$W(x,y,s,\Lambda(s))$ defined in equation 4.5

N.4 Q-Learning - Q(x,s)

1. $B \leftarrow X$

2. $B \leftarrow B \cap \overline{x}$

3. while $|B| > 0$

(a) $y \leftarrow x_i | x_i \in B$

(b) for all $w \in \psi(y, \Lambda(B,s))$

i. $w' \leftarrow \Phi(y,w)$

ii. $Q(x,s,w') \leftarrow \frac{\sum_{z\in\Psi(y,w)} Q(x,s,z)}{|\Psi(y,w)|}$

(c) $B \leftarrow B \cap \overline{y}$

4. $V(x,s) \leftarrow max_a Q(x,s,a)$

VITA

Michael Widener

Candidate for the Degree of

Master of Science

Thesis: ANALYSIS OF SOFT FRIEND OR FOE REINFORCEMENT LEARNING ALGORITHM IN MULTIAGENT ENVIRONMENT

Major Field: Computer Science

Biographical:

Personal Data: Born in Greeley, Colorado, on March 17, 1980.

Education:

Received B.S. degree from Colorado School of Mines, Golden, Colorado, 2002, in Math with Computer Science Option

Completed the requirements for the degree of Master of Science with a major in Computer Science Oklahoma State University in July, 2010.

Experience:

Programmer Intern for Michigan Electronic Materials Company South-West in Sherman, Texas from 2002 to 2003. Software Engineer for Stanley Associates in Lawton, Oklahoma from 2007 to present.

Name: Michael Widener Date of Degree: July, 2010

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: ANALYSIS OF SOFT FRIEND OR FOE REINFORCEMENT LEARNING ALGORITHM IN MULTIAGENT ENVIRONMENT

Pages in Study: 71 Candidate for the Degree of Master of Science

Major Field: Computer Science

This paper evaluates a new off policy multiagent reinforcement learning algorithm called Soft Friend or Foe. The new algorithm is the result of modifying the Friend or Foe [1] algorithm by using the correlation in returns between two agents to soften the distinction between friend and foe. The goal is to achieve results similar to the Nash-Q [3] algorithm without the computational complexity and convergence issues.

Comparison of three multiagent reinforcement learning algorithms is performed on three simple grid world environments. The algorithms consist of: Michael Littman's Friend or Foe algorithm[1], Soft Friend or Foe, and the Q-Learning algorithm[6] adjusted to a multiagent environment.

The Soft Friend or Foe was shown to converge faster than the other two algorithms and get returns equal to or greater than returns received using Q-Learning. Soft Friend or Foe received returns as good as Friend or Foe in all environments.

ADVISOR'S APPROVAL: Dr. Douglas Heisterkamp _____