T.C.

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

A HETEROGENEOUS MULTI AGENT INTELLIGENT PLAYER FOR A REAL-TIME STRATEGY GAME

Mehmet Cihan KURT

THESIS

FOR THE DEGREE OF MASTER OF SCIENCE IN

COMPUTER ENGINEERING PROGRAMME

SUPERVISOR

Assoc. Prof. M. Borahan TÜMER

İSTANBUL 2010

T.C.

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

A HETEROGENEOUS MULTI AGENT INTELLIGENT PLAYER FOR A REAL-TIME STRATEGY GAME

Mehmet Cihan KURT

(141100320060097)

THESIS

FOR THE DEGREE OF MASTER OF SCIENCE IN

COMPUTER ENGINEERING PROGRAMME

SUPERVISOR

Assoc. Prof. M. Borahan TÜMER

İSTANBUL 2010

ACKNOWLEDGEMENT

I want to thank my family and my friends for their endless support and especially Borahan Tümer for his guidance and patience.

TABLE OF CONTENTS

ÖZET

GERÇEK ZAMANLI STRATEJİ OYUNU İÇİN HETEROJEN ÇOKLU ETMENLER KULLANAN AKILLI OYUNCU

Gerçek Zamanlı Strateji oyunları, karşılıklı mücadeleye dayanan, çeşitli kaynakları etkili kullanarak harita üzerine stratejik bölgeleri kontrol altına almaları ve çeşitli manevralar ile karşısındaki tarafın birim ve yapılarını yok ederek kazanmaya çalışmasına dayanan bir oyun türüdür. Satranç ve GO gibi mücadeleye dayalı oyunlardan farkı, gerçek zamanlı ilerliyor oluşudur; oyuncular hamle yapmak için birbirlerinin hamlelerini beklemezler. Oyunun gerçek zamanlı yapısından dolayı, karmaşıklığı karşılaştırılabilir diğer mücadeleye dayalı oyunlara göre çok daha fazladır. Son yıllarda yapay zeka alanında gerçek zamanlı strateji oyunlarına ilgi oldukça artmıştır. Bunun belli başlı nedenlerinden biri, oyunun durum uzayının büyüklüğüdür. Bu tür bir oyun için verimli bir oyuncunun tasarlanması ve geliştirilmesi, durum uzayının büyüklüğünden dolayı verimlilik açısından problemler yaratmaktadır. Bu çalışma, büyük durum uzayı problemine karşı pekiştirmeli öğrenme çerçevesinde bir çözüm üretmektedir. Bu çözümün en önemli katkısı, zamana yayılmış aksiyonlar, doğrusal işlev yaklaşıklamaları ve Yarı-Markov Karar Verme Süreçleri (SMDP) çalışma alanlarını kullanarak durum uzayının genişliğini kabul edilebilir bir seviyeye düşürmesidir. Duruma dayalı karakteristik şablonlar, birime göre özelleştirilmiş aksiyon ve opsiyon şablonları birim bazındaki durum uzayının boyutunu küçültürken, heterojen çoklu etmen kullanımı öğrenmede ayrıştırmayı ve bu sayede farklı stratejilerin durum uzayını büyütmeden ve daha izole biçimde öğrenilmesini mümkün kılmaktadır. Bu çalışma, genel olarak hesaplanabilirlik sınırlarının dışına çıkan bir öğrenme problemini kabul edilebilir verimlilikle gerçekleştirmektedir.

Haziran, 2010 **Mehmet Cihan KURT**

ABSTRACT

A HETEROGENEOUS MULTI- AGENT INTELLIGENT PLAYER FOR A REAL-TIME STRATEGY GAME

A Real-time Strategy (RTS) game is an adversarial game where the participants position and maneuver units and structures under their control to secure areas of the map and/or win by destroying their opponents' units and structures. It is possible to train new units and build new structures during the course of a game with limited amount of resources present in the setting. As opposed to comparable turn-based adversarial games such as chess and go, opponents do not wait for each others' moves so the game progresses naturally with units interacting each other in real time. As a result of the real-time nature of the game, the complexity of the game dramatically increases. In recent years, there has been an increasing interest in RTS games in the artificial intelligence community, especially from the perspective of reinforcement learning, to create an agent to play the RTS game, due to the problem's large state-action space. Modeling and implementing an efficient learning agent that is able to cope with the large state space is a difficult task. This work offers a solution to the problem of large state space in reinforcement learning problems, especially in the complex domain of adversarial real-time strategy games with heterogeneous effectors by using feature based linear function approximation combined with temporally abstract options introduced with the semi-Markov decision processes (SMDP) framework. Context based feature templates and reduced action sets per effector type in a heterogeneous environment greatly reduce the complexity of the learning problem while increasing the efficiency of the convergence of the learning algorithm. This work accomplishes to solve an otherwise intractable problem with an acceptable efficiency.

June, 2010 **Mehmet Cihan KURT**

LIST OF SYMBOLS

$\boldsymbol{\theta}$: Feature vector

Φ : Feature space

$\boldsymbol{\eta}$ *:* Discount factor

$\boldsymbol{S}$: State (set of states)

$\boldsymbol{A}$: Action (set of actions)

O : Options (set of options)

$\boldsymbol{Q}$ *:* Q-Value (value function of actions)

$\widehat{\boldsymbol{Q}}$: Average Q-Value

$\boldsymbol{\pi}$: Policy

ABBREVIATIONS

RL : Reinforcement Learning

MDP : Markov Decision Processes

SMDP : Semi Markov Decision Processes

TD Learning : Temporal Difference Learning

LFA : Linear Function Approximation

LIST OF FIGURES

PAGE NUMBER

LIST OF TABLES

CHAPTER I.

INTRODUCTION

The ability to learn to act rationally (Russell & Norvig, 2003) is one of the oldest challenges of the field of artificial intelligence (AI). How humans are able to learn to act rationally is also an important question for other sciences and fields such as philosophy, psychology and neuroscience. Designing and implementing an agent that is able to form abstractions and learn a specific task to execute it efficiently has attracted a lot of researchers in various different fields. There has been enormous progress both in theoretical studies and domain applications for agents that are able to learn. Applications include cleaning robots that are able to discover their surroundings, sweep the whole floor then go to the docking station to recharge (Russell & Norvig, 2003) or computer agents that are able to beat human world champions in chess (Hsu, 2002) or backgammon (Tesauro, 1995). IBM's Deep Blue Computer attracted a lot of media coverage and public interest when it managed to beat world chess champion Kasparov in 1997. Gerald Tesauro's backgammon playing system TD-Gammon that utilizes sophisticated reinforcement learning (RL) and neural network (NN) techniques is considered to be playing backgammon at world class.

In recent years, there has been an increasing interest in Real-time Strategy (RTS) games in the AI community, especially from the perspective of RL, due to large state space, large action space and delayed rewards (Russel et al., 2005). A RTS game is an adversarial game where the participants position and maneuver units and structures under their control to secure areas of the map and/or win by destroying their opponents' units and structures. It is possible to train new units and build new structures during the course of a game with limited amount of resources in the environment. As opposed to comparable turn-based adversarial games such as chess and backgammon, opponents do not wait for each others' moves so the game proceeds naturally with units interacting

with each other in real time. RTS is an attractive domain to work with RL because it provides a challenging environment for the construction of stable and fast learning mechanisms in response to extremely large state-action space, multiple effectors, both friendly and adversarial, noise and sometimes non-stationarity existing in the environment.

In this work, we used a simplified RTS with only two types of units: workers and marines. Workers gather minerals from mineral patches, and minerals are consumed to train more workers or marines. Marines are able to combat. Each player has a single base, training of workers or marines are carried out in the base. The goal is to destroy all opponents' units, including the base. The opponent plays with a fixed strategy. The domain is continuous. At each step, the action taken by the learning agent is the composition of all actions taken by the base, workers and marines, so the dimensionality of the action space can increase or decrease over time. Reward is based on the outcome of the game. In the case of a tie, scores are assigned based on the relative accomplishments of both players.

In Figure I-1 - Game Instance an instance of a RTS game is shown. Enemy units are shown in gray. Worker1 is navigating to the Mineral Patch to collect resources, while Worker2 is returning to the base with the mineral patch it collected. Marine1 and Marine2 are attacking Marine3.

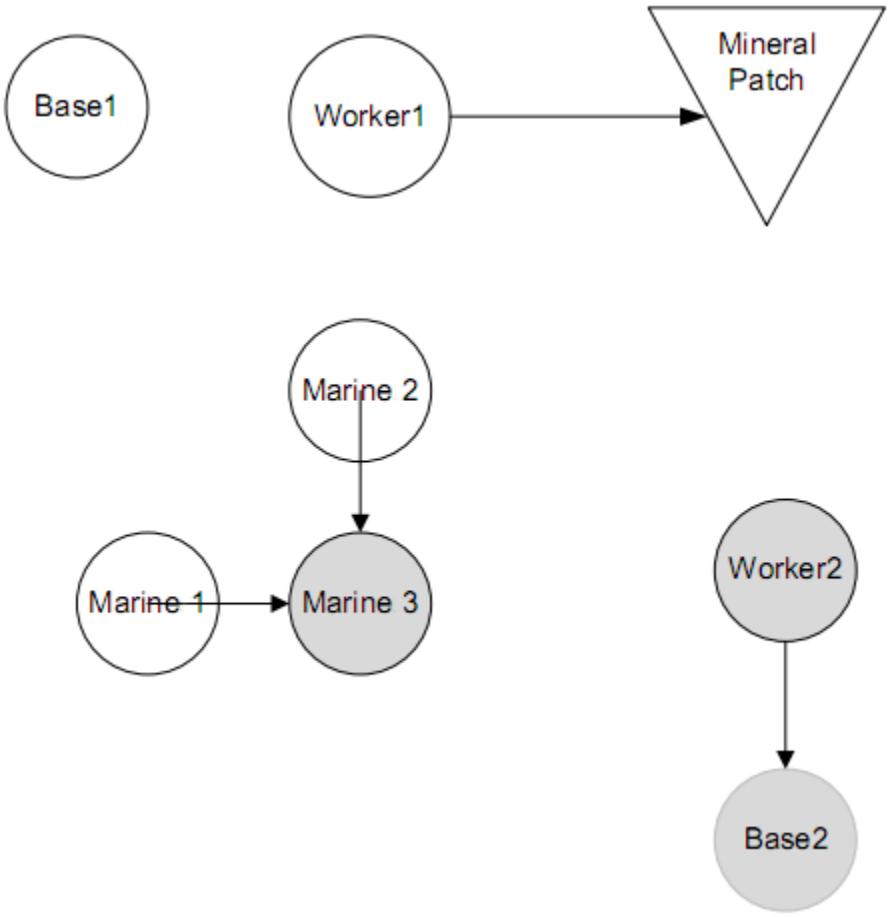

Figure I-1 - Game Instance

There have been some attempts to apply reinforcement learning to RTS games. Ponsen (Ponsen et al., 2006) used a modified version of Hierarchical Semi-Markov Q-Learning (HSMQ), based on MAXQ Value Function Decomposition algorithm (Dietterich, 2000), to show that hierarchical learning helps in a simple task. Madeira used bootstrap learning, with levels of abstraction, to learn parameters one level at a time while using a fixed policy at other levels (Madeira et al., 2004). Ponsen used abstract representations of states and utilized dynamic scripting (Spronck et al., 2006) to update the values of each agent's policy (Ponsen et al., 2006). Sharma used transfer learning to learn in a hybrid of case-based reasoning and RL (Sharma et al., 2007). Some of the works attack the problem by applying a relational model of the effectors to address large-state-space problem by reducing the interaction maps of the effectors (Guestrin & Gearhart, 2003). Other works utilize temporally abstract options (SMDPs) to solve the delayed reward problem and the large-state-space problem (Parr, 1998) (Russel et al., 2005) (Dietterich, 2000). Dietterich has worked on automatic discovery of hierarchies to remove the engineering effort of designing the hierarchy (Dietterich, 2008). There have also been attempts to apply dynamic scripting by using genetic algorithms to generate playing strategies from hard-coded scripts (Sharma et al., 2007) (Madeira et al., 2004). Ghavamzadeh used an average reward scheme instead of discounted reward using hierarchical and recursive policies, in contrary to the general flat policy representations that are usually paired with average reward optimality criterion (Ghavamzadeh & Mahadevan, 2007).

In this work, we aim to offer a solution to the problem of large state space, large action space and delayed rewards in RL problems, especially in the complex domain of adversarial RTS games. The main contributions of our approach are two-fold 1) using a multi-agent structure to model a learning agent with customized action sets and feature templates to reduce the state-space, and 2) constructing a dynamic feature template structure that adapts to the environment and can scale well. We have used effector based feature templates and reduced action sets per effector in a heterogeneous environment to greatly reduce the state-space and action-space of the learning problem while increasing the efficiency. Context based feature templates and reduced action sets per effector type in a heterogeneous environment greatly reduce the state space and action space of the

learning problem while increasing the efficiency. In the resource collection experiment, we focused on feature based linear approximation to show that it reduces the state-space for faster convergence. In the strategic combat experiment, we showed that SMDP framework with options helps in building a simpler, intuitive and hierarchical game model that increases convergence speed by reducing the state-space and enables fair distribution of the delayed rewards. In our game model, effectors are assigned individual tasks so that they are only responsible for the outcome of their own tasks, rewarded by how well they performed and how quickly they execute their tasks. In this experiment we also compared static and dynamic features and showed that it is possible to create agents that can adapt to changing environmental conditions by using dynamic features. In the resource planning experiment, a multi-agent model with hierarchical SMDP structure is constructed. The multi-agent consists of two heterogeneous agents that are specialized in different areas, first agent is the base agent, second is the marine agent. In this experiment we show that separation of agents to form a multi-agent structure lead to better results in convergence and learning.

The rest of this thesis is organized as follows: Chapter II starts with the literature survey about RL applications in the RTS domain. General RL algorithms and their extensions on generalization and temporal abstraction methods, LFA and features are discussed. In Chapter III, learning algorithm is introduced with the model, state representation, actions and rewards. Chapter IV describes experiments and their results ending with discussions. Chapter V concludes the thesis and mentions possible future work.

CHAPTER II.

RELATED WORK

In recent years, there has been an increasing interest in the RTS game domain in RL community. Relational Markov decision processes (RMDPs) (Guestrin & Gearhart, 2003) have been utilized to model the game using interaction maps and relationships of the effectors in the system. These interaction rules and relationships are predefined in the system and they are manually derived and engineered from the general rule set of the game, where pairwise interactions of effectors are described. For example, a worker unit in the domain can only interact with the resources in the environment or with the base. It is not possible to interact with other workers or marines. This kind of relationships simplifies the model of the game and thus limits the state space. In our work, we kept our model simpler by eliminating the need for engineering and fine-tuning of these relationships. Hierarchical abstract machines (HAMs) (Parr, 1998) are used to introduce hierarchy in the system via SMDPs, to model the game, since hierarchy naturally exists in the game. This hierarchical abstraction leads to reduced state space. In this work we also take advantage of natural hierarchy inherent in the game by using temporally abstract actions, but in contrary to Russel's work, we let the agents do the learning on the SMDP level and the basic actions on the MDP level are hard-wired, where the hard-wired solutions are optimal policies that the agents have learned to follow by some prior learning process. Russel further improved the performance of his work (Russel et al., 2005) by using ALisp programming language that allows concurrent execution of the tasks in the system so as to optimize learning. There is a natural concurrency in the game since there are multiple effectors, in our model, we allowed multiple effectors to decide at the same time by modeling the real-time aspect of the game in a discrete time step flow, to enable concurrent and thus faster learning.

Our work differentiates itself from other recent works by implementing a more complicated game scenario and higher number of active concurrent effectors in the game. In his work, Russel used a game scenario with up to 20 effectors concurrent in the game with static feature sets (Russel et al., 2005). In our work we have a game scenario with up to 200 effectors active in the game with dynamic features. Our agent can adapt to changing environmental settings.

II.1 REINFORCEMENT LEARNING AND MARKOV DECISION PROCESSES (MDPs)

In the standard RL model, (Sutton & Barto, 1998) an agent is connected to its environment via observations and actions. On each step of interaction the agent observes its environment and gets some indication, *o*, of the current state, *s*; the agent then takes an action, *a*, to generate an output. The action changes the state of the environment, and the environmental evaluation of this state transition is communicated to the agent through a reinforcement signal, *r*. The agent starts to more frequently take actions that tend to increase the long-run sum of values of the reinforcement signal. It can learn to do this over time by systematic trial and error, guided by a wide variety of algorithms.

In the RL framework, the agent makes its decisions as a function of the environment's states. What we would like, ideally, is a state signal that summarizes past observations in a compact form in such a way that all relevant information is retained. A state signal that succeeds in summarizing all relevant information is said to be Markov, or to have the Markov property and a discrete parameter stochastic process (*s(t); t=0,1,2....*) is said to be a Markov decision process (MDP) if, for any set of points, the current state s_t depends only on s_{t-1} (Eugene & Shwartz, 2001). This obviously should depend on more than the immediate observations, however in some cases the same observations may refer to different state representations based on the past observations. While this situation does not violate the Markov property, it changes its degree i.e. first, second or varying order Markov process.

There are basically two types of RL algorithms: model based and model free (Kaelbing et al., 1996). In model-based case the transition probabilities among states are known or can be constructed using a constructive automaton. However when the model of the environment is not known or hard to build, model-free algorithms are preferred. The most commonly used model-free RL algorithms that we also used in this thesis are Monte Carlo and Temporal Difference (TD) prediction algorithms i.e. SARSA and Q-Learning (Sutton & Barto, 1998).

Monte Carlo methods are ways of solving the reinforcement learning problems based on averaging sample returns Figure II-1. To ensure that well-defined returns are

available, Monte Carlo methods are defined only for episodic tasks. That is, we assume experience is divided into episodes, and that all episodes eventually terminate no matter what actions are selected. It is only upon the completion of an episode that value estimates and policies are changed.

Repeat forever:

1. Choose an arbitrary policy π (e.g. ε-greedy)
2. Generate an episode using π
3. For each pair *s,a* in the episode
    a. Observe reward *R*
    b. Update *Q(s,a)* using average of *R*
4. For each *s* in the episode
    a. $a^* = \arg\max_a Q(s,a)$
    b. Update the π

Figure II-1 Monte Carlo Algorithm (Sutton & Barto, 1998)

TD methods on the other hand do not wait for the completion of an episode. Every time a non-terminal state (s_t) is visited the value of the state ($V(s_t)$) is updated based on the reward signal received. We can update the values of the states by following an on-policy or an off-policy algorithm. SARSA (Sutton & Barto, 1998) which uses an on-policy update follows a policy, π and updates the state-action values of the same policy as illustrated in Figure II-2.

Repeat forever:

1. Choose a arbitrary policy π (e.g. ε-greedy)
2. Choose *a* from *s* using π
3. Repeat for each episode
    a. Take action *a* observe *r*, *s'*
    b. Choose *a'* from *s'* using π
    c. Update *Q(s,a)* using *r*

    $Q(s,a) = Q(s,a)+\alpha[r+\gamma Q(s',a')-Q(s,a)]$

    d. *s=s'; a=a'*
4. Until s is terminal

Figure II-2 SARSA on-Policy TD Control (Sutton & Barto, 1998)

Q-Learning (Sutton & Barto, 1998), on the other hand, is an off-policy TD control method which means it updates the values of the actions of a policy for learning, while it follows another policy in action selection. Q-Learning algorithm is shown in Figure II-3.

Repeat forever:

1. Initialize $Q(s,a)$ arbitrarily
2. Choose an arbitrary behavior policy π (e.g. ε-greedy)
3. Repeat for each episode
    a. Choose a from s using π
    b. Take action a observe r, s'
    c. Update $Q(s,a)$ using r

    $Q(s,a) = Q(s,a)+\alpha[r+\gamma max\ Q(s',a')-Q(s,a)]$

    d. $s=s'$
4. Until s is terminal

Figure II-3 Q-Learning off-Policy TD Control (Sutton & Barto, 1998)

II.2 SEMI-MARKOV DECISION PROCESSES

The action, in MDPs, that is taken at time *t* only affects the observations and reward at time *t+1*. There is no notion of a course of action extended in time. Therefore, in a MDP the state representation has to cover all possible relations and it does not let the system designer implement a temporal abstraction. However temporal abstraction can be added to an MDP framework by extending it to SMDPs.

SMDP is the minimum extension to the RL framework (Bradtke & Duff, 1994). SMDPs are a special kind of MDP modeled for continuous time discrete event systems. In an SMDP an action taken at time *t* does not have to be evaluated by the main system at *t+1*. Temporal abstraction provides system designers with evaluation of the action at a different time scale without intervening it at the main framework. Therefore in an SMDP one can use temporarily extended course of actions instead of primitive action selection-evaluation method. Figure II-4 shows the relationship of actions and options in MDP, SMDP and Options over MDP.

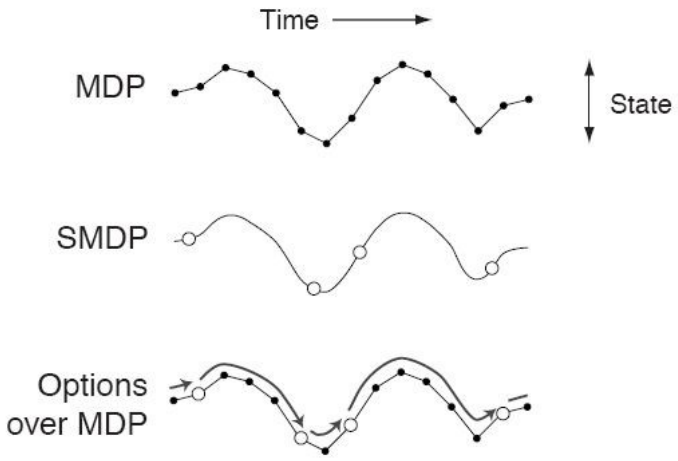

Figure II-4 MDP, SMDP, Options over MDP (copied from (Sutton et al., 1999))

Sutton called this temporarily extended course of actions as *options* as illustrated in Figure II-5 and re-designed them by allowing *intra-option modeling* (Sutton et al., 1999). In SMDP options are accepted as an indivisible deterministic sequence of actions. There is no attempt in SMDP theory to look inside the temporally extended actions, to examine or modify their structure in terms of lower-level actions. However one can learn and optimize inside of options by modeling and giving appropriate reward signals to lower level actions at different time scales.

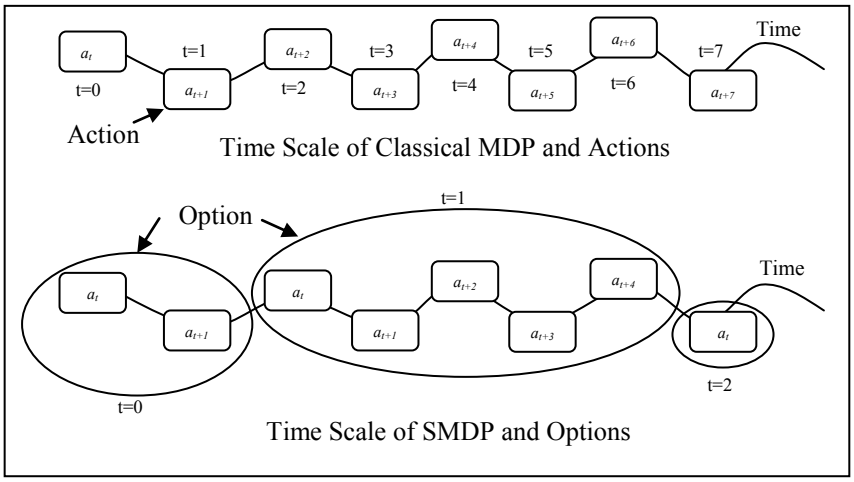

Figure II-5 Action and time models in MDP and SMDP

In the SMDP framework, Q-Value for the option state pairs follows directly from the MDP case as can be seen in the formula, options replacing actions:

$$Q(s,o) = Q(s,o)+\alpha[r+\gamma \max Q(s',o')-Q(s,o)] \qquad \text{(II-1)}$$

SMDPs have been extensively used in RL research especially in modeling RL problems with temporally abstract options and hierarchical abstractions to reduce state-space and solve the delayed reward problem (Parr, 1998) (Russel et al., 2005) (Dietterich, 2000) (Guestrin & Gearhart, 2003).

II.3 FUNCTION APPROXIMATION

Finding a solution to an MDP problem in RL framework requires functional mappings such as value functions, Q-functions, and policies. While these functions are simply represented by lookup tables which are easy to operate on, this simple representation scheme is only limited to applications with a small state-action space. In real world problems such as a RTS game or continuous time/ continuous state Markov processes, the extremely large number of state-action pairs leaves the most of the state-action pairs not sufficiently experienced and hence requires a different way to represent the functions (e.g. The values *Q(s,a)* of state-action pairs); so that the agent may generalize from the instances sufficiently experienced to those not ever encountered. The first and the most basic way of representing such functions is using a lookup table, which is linear in the size of the domain in both memory and computational costs. The computational complexity grows linearly with the state space or the cartesian product of the state space and action space.

Linear dependency is infeasible for domains with large state spaces, such as a RTS game. Even our simplified RTS game has over 2^{100} states, it can reach up to 2^{500} states in a regular RTS game. It can be the case that functions have a special structure allowing them to be represented in a compact form. Function approximation helps in representing the Q-Value of a state in a more compact form, this helps reducing the state space size because the real world state is represented in a more compact form that is smaller in size thus enabling efficiency.

The sort of generalization that we need to incorporate to our system is called *function approximation*. In a Q-function learning algorithm, for example, the designer might specify a set of Q-functions $\{Q_\theta (s,a) \mid \theta \in \Theta\}$ where Θ represents a parameter space in which the Q-function is expected to be approximated by a member of this set of Q-functions. Most of the RL algorithms can work on the approximated representation with the optimal member of Θ. The tabular or lookup table representation is the simplest way of approximating a function. There is a parameter $\theta_{s,a}$ for each state s and action a, and $Q_\theta (s, a) = \theta_{s,a}$. To generalize, we assume that certain state–action pairs have the same Q-value, so we can construct a function ϕ from $S \times A$ to some *feature space* Φ, that

linearly maps these certain pairs to the same Q-value. In the end, we have a function approximator with a parameter space consisting of all functions from Φ to R, and Q-value is calculated as $Q_\theta(s, a) = \theta(\phi(s, a))$. This is called the state-aggregated tabular approximation (Sutton & Barto, 1998).

Function approximators basically lead to the generalization of states and actions. The state-aggregated tabular approximation generalizes between state–action pairs meaning that if they have to the same value in the feature-space, then they have to have the same Q-value. A more flexible form of generalization is possible by linear approximation, which can be summarized as a mapping $\phi : S \times A \rightarrow \mathrm{R}^{\mathrm{M}}$. Each component $\phi_{\mathrm{m}}(s, a)$ is called a feature, and $\phi(s, a)$ is called a feature-vector. The parameter space is then R^{M}, and the corresponding Q-functions can be represented as follows:

$$Q_\theta(s, a) = \theta^{\mathrm{T}}(\phi(s, a)) \qquad \text{(II-2)}$$

In the next chapter we will be explaining the general model of our agent, the environment, the internal hierarchy and the general multi agent structure with details about the feature, action and option structure.

CHAPTER III.

LEARNING

Our simplified RTS game environment consists of units interacting with each other in the map in real time. Time is discrete, so a new action is taken by both of the opponents at each time step and the actions take effect immediately at the same time. The 2 dimensional coordinate system of the map is discrete so units move in discrete steps and directions.

There are 3 types of units in the environment. *Base* is the headquarters, it is unable to move. An opponent can only have a single base at the same time. *Base* is where the training of new units and storage of the collected resources take place. *Worker* is the unit that can move freely in the map and can gather resources from mineral patches. *Marine* is the only unit that is armored with fire power. It can fire at and kill other units, as it is able to move freely in the map.

III.1 MODEL OF THE GAME

A real-time strategy game is an adversarial game where players try to beat each other within a real-time environment by generating intelligent or hard-coded strategies. Our simplified realtime environment consists of base, worker, marine units and mineral patches. Bases are headquarters for the players, workers collect mineral patch that can be used to train other workers or marines. Workers do not have any weapon. Marines are military units that are able to attack and fire. Mineral patches are static locations in the map with limited amount of resources and these resources can be collected by any player. The map is limited to a rectangular area and coordinates are discrete. The game progresses in discrete time intervals, both players take decisions in every interval and it is applied immediately in the next time interval so there is no induced advantage. When all the units of a player are destroyed or maximum allowed time for an episode is reached the episode ends. In Figure III-1 RTS Reinforcement Learning Model our learning model is shown where the agent observes the state of the environment, then take actions and after that, the environment rewards the agent. Our RL model focuses on modeling the state of the environment which the agent observes. Agent consists of multiple agents consisting of base, marine and worker agent. There is a two level hierarchy, in the MDP level actions are learned, in the SMDP level options are learned. Actions in the MDP level are single time step actions, to keep our model simple and to focus on the SMDP aspect of the game, policy at the MDP level is hard-wired. LFA, static features and dynamic features are utilized for better performance.

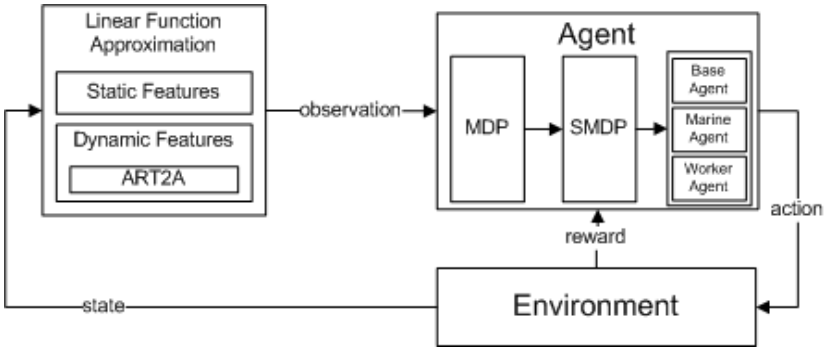

Figure III-1 RTS Reinforcement Learning Model

III.1.1 Features

In function approximation, it is a challenging task to come up with a sound compact representation of the world state that is both simple enough to be efficiently calculated and complex enough to give a good all around representation of. Features are usually constructed with specific domain knowledge, so an expertise in the domain is preferred for good results.

In his work, Ponsen uses the same RTS domain as we use in this work, in a relatively simple task (Ponsen et al., 2006). A worker tries to navigate to a resource to collect it, while avoiding an enemy marine that can kill it. Ponsen uses 4 features for a compact representation of the world. Distance(enemy,s), Distance(goal,s), DirectionTo(enemy,s) and DirectionTo (goal,s). The function Distance returns a number between 1 and 8 or a string indicating that the object is more than 8 steps away in state s, while DirectionTo returns the relative direction to a given object in state s. Using 8 possible values for the DirectionTo function, namely the eight directions available on a compass rose, and 9 possible values for the Distance function.

In our work, we utilized several features to be able to capture a sound and compact representation. In order to reduce the state space, states related to objects outside of the visibility range of a marine or a worker are considered as irrelevant and not taken into account. For the sake of simplicity, all our features are one dimensional binary vector. In Table III-1, all features present in the feature space are listed. In our literature survey we did not come by a work that used the feature set we have utilized in our work. Most of the works used features using distance, angle for modeling the relationship of the effectors.

Feature name	Value
$f_{0\ Enemies}$	1 if 0 enemy units are in visible range, 0 otherwise
$f_{1\ to\ 3\ Enemies}$	1 if 1 to 3 enemy unit is in visible range, 0 otherwise
$f_{4\ to\ 7\ Enemies}$	1 if 4 to 7 enemy units are in visible range, 0 otherwise
$f_{8\ to\ 12\ Enemies}$	1 if 8 to 12 enemy units are in visible range, 0 otherwise
$f_{13\ to\ 18\ Enemies}$	1 if 13 to 18 enemy units are in visible range, 0 otherwise
$f_{19\ to\ 25\ Enemies}$	1 if 19 to 25 enemy units are in visible range, 0 otherwise
$f_{26\ or\ more\ Enemies}$	1 if 26 or more enemy units are in visible range, 0 otherwise
$f_{0\ Friends}$	1 if 0 friendly units are in visible range, 0 otherwise
$f_{1\ to\ 3\ Friends}$	1 if 1 to 3 friendly unit is in visible range, 0 otherwise
$f_{4\ to\ 7\ Friends}$	1 if 4 to 7 friendly units are in visible range, 0 otherwise
$f_{8\ to\ 12\ Friends}$	1 if 8 to 12 friendly units are in visible range, 0 otherwise
$f_{13\ to\ 18\ Friends}$	1 if 13 to 18 friendly units are in visible range, 0 otherwise
$f_{19\ to\ 25\ Friends}$	1 if 19 to 25 friendly units are in visible range, 0 otherwise
$f_{26\ or\ more\ Friends}$	1 if 26 or more friendly units are in visible range, 0 otherwise
$F_{Am\ I\ Being\ Attacked}$	1 if the unit is under attack, 0 otherwise
$f_{Low\ Health}$	1 if the unit health is below 50 percent, 0 otherwise
$f_{Friend\ Under\ Attack}$	1 if a friendly unit in visible range is under attack 0 otherwise
$f_{0\ Workers}$	1 if agent has 0 workers, 0 otherwise
$f_{1\ to\ 3\ Workers}$	1 if agent has 1 worker, 0 otherwise
$f_{4\ to\ 7\ Workers}$	1 if agent has 2 workers, 0 otherwise
$f_{8\ to\ 12\ Workers}$	1 if agent has 3 workers, 0 otherwise
$f_{13\ to\ 18\ Workers}$	1 if agent has 4 workers, 0 otherwise
$f_{19\ to\ 25\ Workers}$	1 if agent has 5 workers, 0 otherwise
$f_{26\ or\ more\ Workers}$	1 if agent has 6 or more workers, 0 otherwise

Table III-1 Features

III.1.2 Actions

In a RL problem, the main objective is to optimize action decisions over the state space. In our simplified RTS game domain, there are 3 kinds of units that have different action spaces. For example, while a marine is able to execute “attack” action over other units, base and workers do not have this action in their action spaces. While some of the units share common actions, separation of action spaces per unit leads to a reduced action space. The only action that is shared among all unit types is the Noop (No operation) action.

Base is distinctively different from other 2 unit types. Base cannot move. It can train new units as necessary, a new marine or a new worker unit.

Worker can only move in 4 different directions, up, down, left and right. It is able to pick resources from the mineral patch and drop resources at the base, but for the sake of simplicity, the pick-up and drop actions are removed from the action space. When the worker is at a mineral patch, it automatically does the pick-up action, and when it is at a base, it does the drop action.

Marine is able to move and attack other units. It is the main fighting unit that is able to destroy enemy units. Its action space consists of 4 different directions, up, down, left, right and an attack action that targets another unit. If the marine executes the attack action, health of the targeted unit decreases.

In Table III-2, units and the actions that they can take are listed.

Action \ Unit	Base	Worker	Marine
Move Up	-	+	+
Move Right	-	+	+
Move Down	-	+	+
Move Left	-	+	+
Pick-up Resource	-	+	-
Drop Resource	-	+	-
Train Worker	+	-	-
Train Marine	+	-	-
Attack	-	-	+

Table III-2 Action-Unit Matrix

III.1.3 Options

In the SMDP framework, options are defined as temporally abstract actions (Sutton et al., 1999). In our model, options are tasks that usually take more than one discrete time step to execute. Units execute these options by taking a sequence of actions that span over time. For example a "navigate" task assigned to a worker to go from the base unit to the mineral patch can only be executed by a sequence of actions up, right, down and left. Base executes the "train worker" task by taking the train worker action that takes 10 time steps. Marine executes the "attack" task by taking the "attack" action on each time unit until the unit itself or the enemy unit it is attacking dies. A task can take any amount of time step until it reaches the defined terminal state of the task.

Name	Definition	Terminal State
$O_{EXPLORE}$	Take one of the up, right, down, left actions randomly	Unit is attacked or 5 time steps have passed
$O_{ATTACK\text{-}CLOSEST\text{-}ENEMY}$	Pick closest enemy in visible range and attack	Attacking unit or the enemy is dead
$O_{ATTACK\text{-}WEAKEST\text{-}ENEMY}$	Pick weakest enemy in visible range and attack	Attacking unit or the enemy is dead
O_{NOOP}	Do nothing	Single time step
O_{FLEE}	Navigate to a point where no enemy units are in visible range	Unit is in a location where no enemy units are in visible range or unit is dead
$O_{NAVIGATE\text{-}TO\text{-}WEAKEST\text{-}FRIEND}$	Navigate to a friend that is weakest in the visible range.	Until destination point is reached or unit is dead
$O_{NAVIGATE\text{-}TO\text{-}CLOSEST\text{-}FRIEND}$	Navigate to a friend that is closest in the visible range.	Unit destination point is reached or unit is dead
$O_{NAVIGATE}$	Navigate to a destination point	Unit destination point is reached or unit is dead
$O_{TRAIN\text{-}WORKER}$	Train a worker	10 time steps have passed or unit is dead.
$O_{TRAIN\text{-}MARINE}$	Train a marine	10 time steps have passed or unit is dead.

Table III-3 Options

Options are specifically constructed so as to enable the units to take strategic decisions, for example O_{FLEE} option is to flee from the enemy, so a unit can learn to flee from an enemy unit if its health is below a threshold. $O_{ATTACK\text{-}CLOSEST\text{-}ENEMY}$ is an option that picks the closest enemy unit in visible range and starts attacking it. $O_{ATTACK\text{-}WEAKEST\text{-}ENEMY}$ is an option that picks the weakest enemy unit in visible range and starts

attacking it. These two different options with minor differences can create great differences in the outcome of the fight between a few marines. For example, $O_{\text{NAVIGATE-TO-CLOSEST-FRIEND}}$ which enables a marine to immediately go to a friendly unit for help can mean the difference between life and death for that friendly unit.

Option space is different per unit type, for example, while O_{EXPLORE} is enabled for workers and marines, base does not have this option since it is unable to move. It is important to emphasize that options are strictly a sequence of actions so actions are their only building blocks. In Table III-4, units and their options spaces are listed. In the literature survey we conducted we did not come by option usage types such as ours in the works related to RTS games.

Option \ Unit	**Base**	**Worker**	**Marine**
O_{EXPLORE}	-	+	+
$O_{\text{ATTACK-CLOSEST-ENEMY}}$	-	-	+
$O_{\text{ATTACK-WEAKEST-ENEMY}}$	-	-	+
O_{NOOP}	+	+	+
O_{FLEE}	-	+	+
$O_{\text{NAVIGATE-TO-WEAKEST-FRIEND}}$	-	-	+
$O_{\text{NAVIGATE-TO-CLOSEST-FRIEND}}$	-	-	+
O_{NAVIGATE}	-	+	+
$O_{\text{TRAIN-WORKER}}$	+	-	-
$O_{\text{TRAIN-MARINE}}$	+	-	-

Table III-4 Option-Unit Matrix

III.1 LEARNING

In the RL framework, there are two variants of temporal difference (TD) learning algorithms: Q-Learning and SARSA. It is possible to categorize learning algorithms as *on-policy* algorithms and *off-policy* algorithms (Sutton & Barto, 1998). On-Policy algorithms use the same policy to both select actions from and update their values, while off-policy algorithms keep a policy for update and another policy to follow for action selection.

There are different types of algorithms for computing the optimal policy based on TD. Q-learning calculates an estimate $\hat{Q}$ of the optimal Q-function. For an observed transition (*s, a, r, s'*), it performs the update

$$\hat{Q}_\theta(s, a) \leftarrow \hat{Q}_\theta(s, a) + \eta(r + max_{a'} \hat{Q}_\theta(s', a') - \hat{Q}_\theta(s, a)) \quad \text{(III-1)}$$

There is an exploration vs. exploitation tradeoff related to reward maximization (Sutton & Barto, 1998), defined by a parameter epsilon, the probability of random action, instead of choosing the action that maximizes the Q value. This random exploration is necessary so that the value is not stuck in a local minima (Sutton & Barto, 1998). It is possible to incorporate function approximation into TD algorithms. Let's assume that the Q-function is approximated as $Q_\theta(s, a)$ where the parameter θ is in R^M. Then the Q-update can be formulated as below where $\nabla Q_\theta(s, a)$ represents the delta:

$$\theta \leftarrow \theta + \eta(r + max_{a'} Q_\theta(s', a') - Q_\theta(s, a)) \nabla Q_\theta(s, a) \quad \text{(III-2)}$$

The case with the function approximation is that, instead of the Q-function, the parameter vector is being updated and there is an extra gradient termt at the end. Actually, parameter vector is what is being learnt. In this work we use linear function approximation so the gradient term at the end simply reduces to a feature vector and the resulting update formula can be summarized as

$$\theta \leftarrow \theta + \eta(r + max_{a'} Q_\theta(s, a) - Q_\theta(s, a)) \vec{f}(s,a) \quad \text{(III-3)}$$

where $\vec{f}$ is the feature vector. Then the Q-Value is calculated as

$$Q_\theta(s, a) = \sum_{k=1}^{K} \theta_k f_k(s,a) \quad \text{(III-4)}$$

III.1.1 Rewards

In most of the RL domains, reward is immediate. It is received from the environment just after the action is taken so it is relatively easy to know which action gives the best reward for a state; after all possible actions are taken for all states. However, in the RTS game domain, the outcome of the game, that is the reward, is only known at the end of the game, when either of the opponents is destroyed or maximum number of steps allowed for an episode is reached. It is a rather challenging task to distribute the reward taken at the end of the episode to the intermediate actions taken during the episode.

In the SMDP framework, all reward accumulated by actions taken within the temporal span of the option is the total reward for that option. Q-Value update is done at the terminal state of the option. In our game model, there can be a single base for each opponent while there is no limit to the number of marine or worker an agent can have at any given state. This leads to concurrent options being executed by any number of marines and workers. For example, a marine can make a decision to execute "attack" option on an enemy unit at time t_0, and at time t_5 this marine is dead. Total reward at the end of that option is the total number of time steps times the -1 reward that is given per time step for all units added by the -5 penalty for being killed at the end of an "attack" option. Assume that this same decision was taken concurrently by 5 marines and some of them are killed in the middle of the attack option and some of them survive by killing enemy marines. All concurrent options accumulate their own rewards and a single marine is updated at the end. The term $\vec{f}(s,a)$ at the end of the update function is calculated as a difference of the feature vector at time t_n and t_0 for that specific unit that can be formulated as

$$\vec{f}(s,a) = \vec{f}_{t_n}(s,a) - \vec{f}_{t_0}(s,a) \quad \text{(III-5)}$$

Q-Value update is done per unit option and $\vec{f}(s,a)$ that is used in this update is calculated per unit by saving feature vector at the beginning of the option and at the end of the option.

General rewarding rule in our model is; -1 penalty is given at each time step to all units, it motivates the units to complete the option they are in as soon as possible. Marines get a -5 penalty if they are killed in an attack and +5 reward if they survive an attack option by killing the enemy marine. At the end of the game, a general reward is received and it is distributed evenly to all units alive in the game. General reward that is received at the end of the game is r_{gen} . No reward is given if the agent loses the episode.

$$r_{gen} = \begin{cases} 100 - (total\ time\ steps\ of\ the\ episode) & \text{if agent wins} \\ 0 & \text{if enemy wins} \\ distribute(100 - total\ time\ steps\ of\ the\ episode) & \text{if draw} \end{cases} \quad \text{(III-6)}$$

III.2 HETEROGENEOUS AGENT MODEL

We designed a heterogeneous multi-agent model that consists of two learning agents that work in cooperation. In order to achieve better separation and abstraction, we implemented two different agents that are functionally different, first one the base learning agent that learns the general behavior of an intelligent base, and the second one is the marine learning agent that learns how to fight enemy units in groups. There is a third unit that could have been also modeled as a learning agent but worker agent's task is quite simple, a resource collection task that consists of navigation between two map coordinates. Workers follow a fixed policy of resource collection task except in the first experiment in the experiments section that represents a simple resource collection task while avoiding an enemy unit. In Figure III-2, please notice that marines concurrently taking decisions and return the accumulated rewards for the Q-Value update.

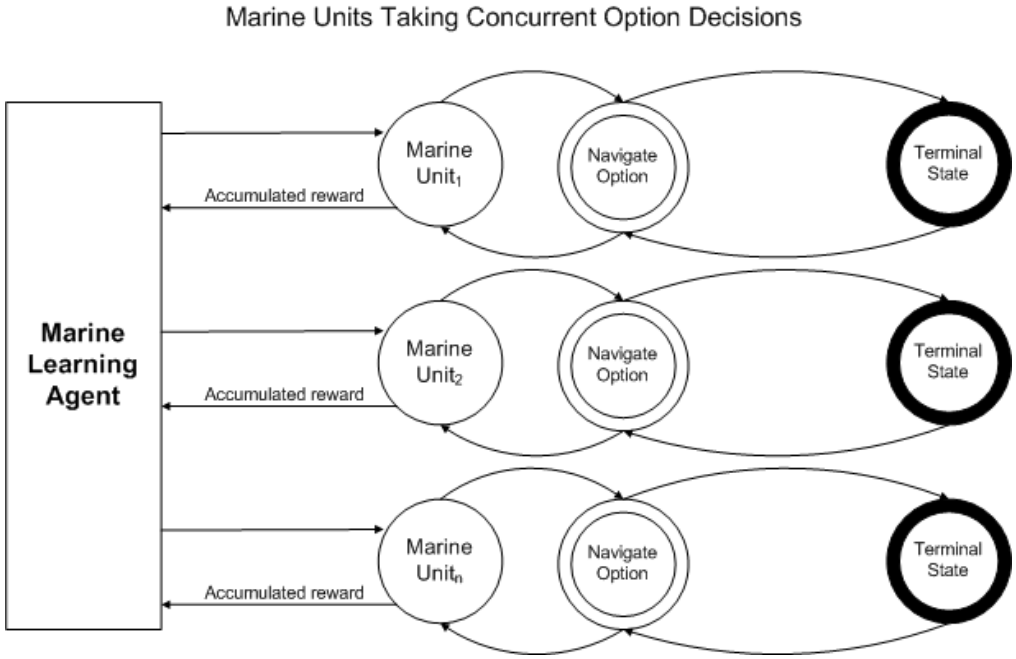

Figure III-2 Marine Units Taking Concurrent Option Decisions

Base and marine learning agents are not aware of each other and they do not communicate explicitly, they just observe the portion of the real world state that they

need through feature vectors. As we have mentioned in the previous sections, base and marines have different option and feature spaces. This specialization and separation of agents, also include separation of rewards, so a base agent does not get rewards for a successful marine agent that kills enemies, every agent is responsible for its specialized task. In Figure III-3 hierarchy of options and actions are represented.

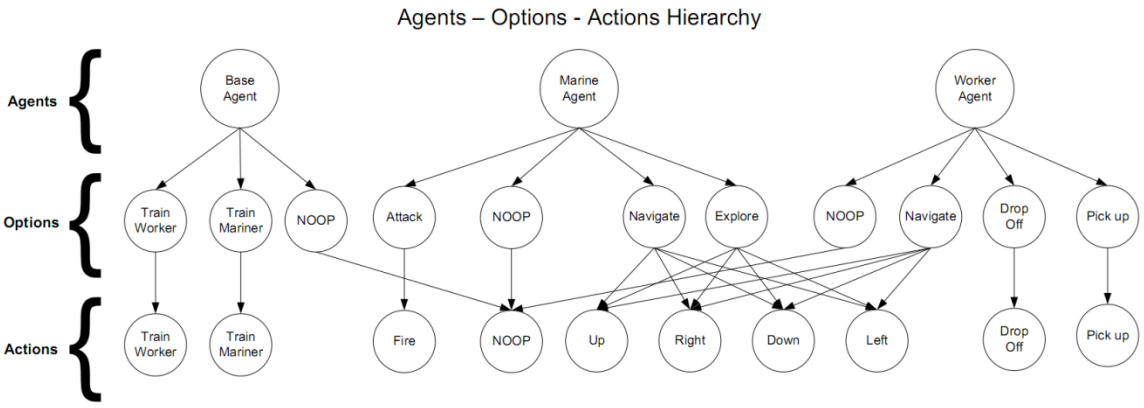

Figure III-3 Hierarchy of options and actions

CHAPTER IV.

EXPERIMENTS AND RESULTS

We experimented with different problems ranging from simple resource collection problem to more complicated war scenarios. The first experiment is a resource collection problem of a worker with an enemy unit in the map that can kill the worker if it is in the visible range. The second experiment is a strategic marine fighting problem where two opponents fight each other with same number of marines. In the third experiment we study the full war scenario with bases, resource collection and unit training.

IV.1 RESOURCE COLLECTION PROBLEM

Resource collection problem is about a worker trying to reach a specific point in the map with resources, namely a mineral patch, while an enemy marine sits in the middle of the map. Worker should be able to learn how to avoid the enemy marine unit while reaching the mineral patch.

There are a few challenges in this problem. First, it is important to design a sound and robust feature set for this specific problem. Second, finding a solution to the problem that generalizes well for problems alike, for example what will happen if the location of the enemy or the location of the mineral patch in the map changes? Using features and linear approximation helps in reducing the state space while choosing a good feature set might help in generalization.

In this problem, grid size is 10 by 10, with 100 distinct positions. Enemy is in the middle of the grid, at position (5, 5). Worker starts at position (1, 1) and goal state is at position (10, 10). The single feature that is used for linear approximation is the

Manhattan distance between the worker itself and the goal. Agent gets a -1 penalty for each time step, reward is 0 for the goal state. A -10 penalty is given if the worker is in the visible range of the enemy marine unit.

Distance feature reduces the state space from 100 to 10. In Figure IV-1, performance of Linear Function Approximation (LFA) (II-2) and Flat Q-Learning (III-1) is compared with number of episodes and the number of steps needed to satisfy the termination conditions. It is obvious that LFA outperforms Flat Q-Learning in the time required for convergence to the optimal policy.

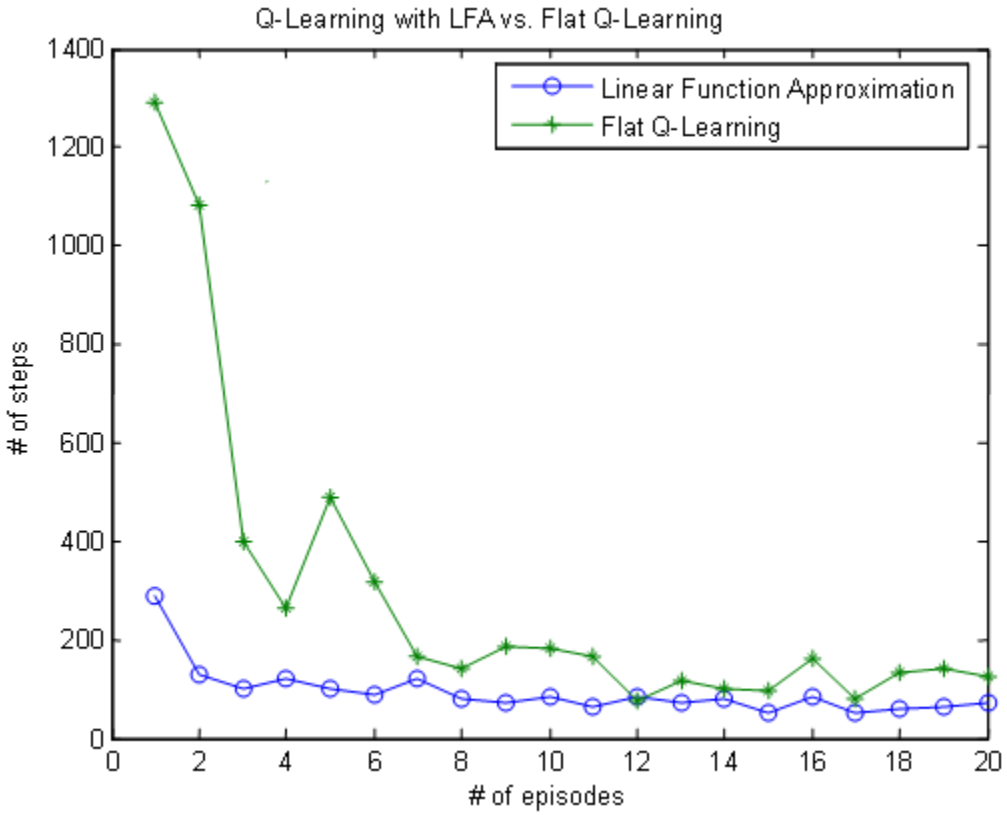

Figure IV-1 Q-Learning with LFA vs. Flat Q-Learning

In the previous experiment we only used a single *distance* feature for state reduction. Now we will add another feature, Angle with the enemy marine unit, this will give extra information to the worker so that it can avoid the enemy unit better.

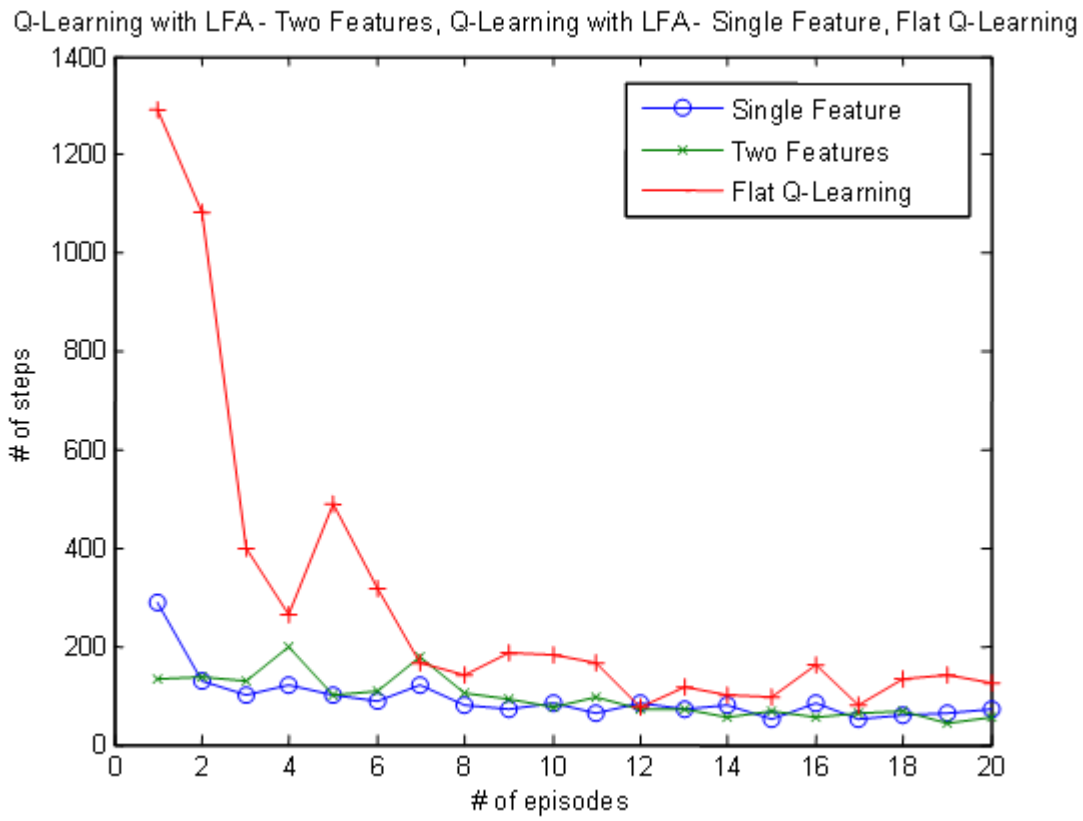

Figure IV-2 Q-Learning with LFA - Single Feature, Q-Learning with LFA - Two Features vs. Flat Q-Learning

In Figure IV-2 , comparison of the LFA with single feature with LFA with two features and Flat Q-Learning can be seen. Two features easily surpass two other algorithms while also converging to the optimum policy more consistently; notice the lower number of spikes in the graph for LFA Two Features.

IV.2 STRATEGIC COMBAT PROBLEM

In the strategic combat problem, our main objective is to allow marine units to learn how to take strategic decisions while fighting in groups against other enemy marine groups. For example take a group of 4 marines confronting a group of 4 marines. If they attack randomly to an enemy marine they choose, it would not be a wise choice since the fire power will be distributed among enemy units and it will take much longer to kill them. If they can somehow figure out to attack the same enemy marine all together, which enemy marine should they choose? In this case, it would be wise to attack the enemy marine with the weakest health level. Through sound and good feature set selection, we are trying to enable marines act rationally in different conditions with any number of friendly marines and enemy marine units in their visible range. Our epsilon value is 0,01 and alpha value is 0,01.

In Table III-2, feature set for a marine is provided. In this list there are features chosen to enable a compact representation of a number of friendly units and enemy units in the visible range. In this problem, marine learning agent uses these features to be able to take rational decisions in different situations and scenarios. In this experiment, we used a 20 by 20 grid, one team starting at position (1, 1), enemy team starting at position (20, 20). Opponents always start with the same number of marines so they fight under the same circumstances. In the beginning, each marine is out of other marines' visibility ranges so marines explore the map until they see an enemy unit. Enemy marines always follow a fixed policy; explore the map randomly until they see an opponent unit, and start attacking when it is in their visibility range. This policy seems like a straightforward and simple policy but it is quite aggressive and effective in fact, note that in our experiment result in Figure IV-3, the optimum policy never converges to 100 percent winning ratio, the enemy always has a 10 percent winning or not losing ratio. In the literature, hard-coded advanced strategies developed by humans also state that aggressive and simple algorithms are quite effective (Ying-zi & Ming-yang, 2003) (Lee et al., 2008). Our agent can only fight against this fixed policy if it is able to employ strategies as a whole team; attacking the same enemy unit at the same time, attacking the weakest enemy unit at first, fleeing if it is close to death.

In this experiment, we started both opponents with 5 marines, increased the number of initial marines to 20 and then to 100. In Figure IV-3, the graph of percentage of not losing is plotted against the number of marines at the episode start. It is possible to observe the differences in convergence time for each experiment. Our static features have been designed for ranges changing from 0 to 25. As you can see in the Figure IV-3, 5 vs. 5 perform worse than 20 vs. 20 case 100 vs. 100 case also performs worse than 20 vs. 20 case. The reason for that performance results is ranges for the static features are chosen for marine numbers ranging from 0 to 25 on average. In the next section, dynamic features, we will be designing our features dynamically so it will be able to adapt to the change.

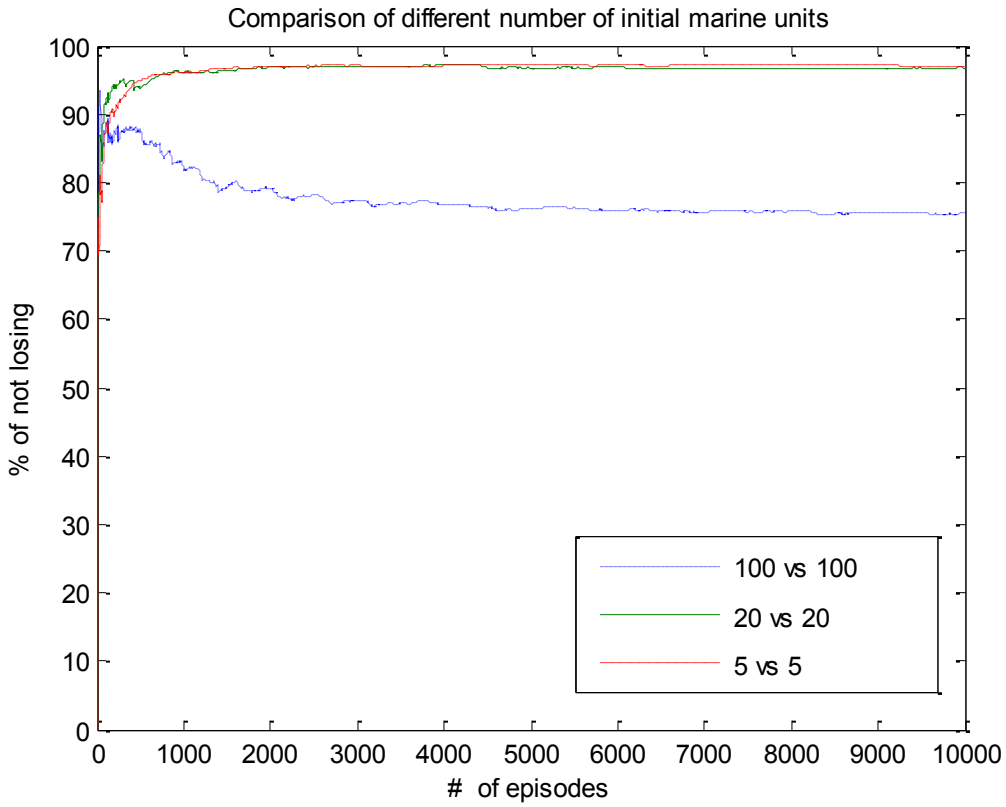

Figure IV-3 Comparison of different number of initial marine units

IV.3 Dynamic Features

As we have discussed in the previous section, Feature 1 and Feature 2, which are the number of enemies around and the number of friends around have static ranges, which are selected by engineers' best guess. The ranges for clustering in the static feature are [0], [1:3], [4:7], [8:12], [13:18], [19:25], [26 or more]. However, when the number of units in the problem changes, or the problem is moved to a different domain, these static selections might cause problems. Supervising these changes takes time and finding the optimum values is a challenging task. This is known as the generalization problem (Guestrin & Gearhart, 2003) (Ponsen et al., 2006). For this reason, we tried to create an adaptive dynamic feature structure, so that these ranges are selected during the game play. ART2a neural networks are able to cluster the sequence of inputs (Carpenter et al., 1991). The observation sequence of number of enemies or friends around form the input to the ART2a network and a clustering is performed during the game play. For example, a scenario with 5 marines vs. 5 marines creates a different clustering while a scenario with 100 marines vs. 100 marines creates a totally different clustering. Figure IV-4 and Figure IV-5 show the performance comparison of static Feature 1 and dynamic feature with ART2a that represents "number of enemies around" feature for 5 vs. 5 marines and 100 vs. 100 marine case. In the 5 vs. 5 marines case there is no notable performance difference between the applications. However, in the 100 vs. 100 marine case, dynamic feature with ART2a outperforms static feature. This performance rise is due to the adaptive nature of the dynamic feature sets, it clusters the observation for Feature 1. Static features are unable to capture the difference between the environment with lower and higher number of enemy units so when the number of enemy units rise, they perform worse than dynamic feature sets. Static and dynamic features in the 5 vs. 5 marines case converge to the same value. Note that dynamic features perform dramatically better than static features in the 100 vs. 100 marine case.

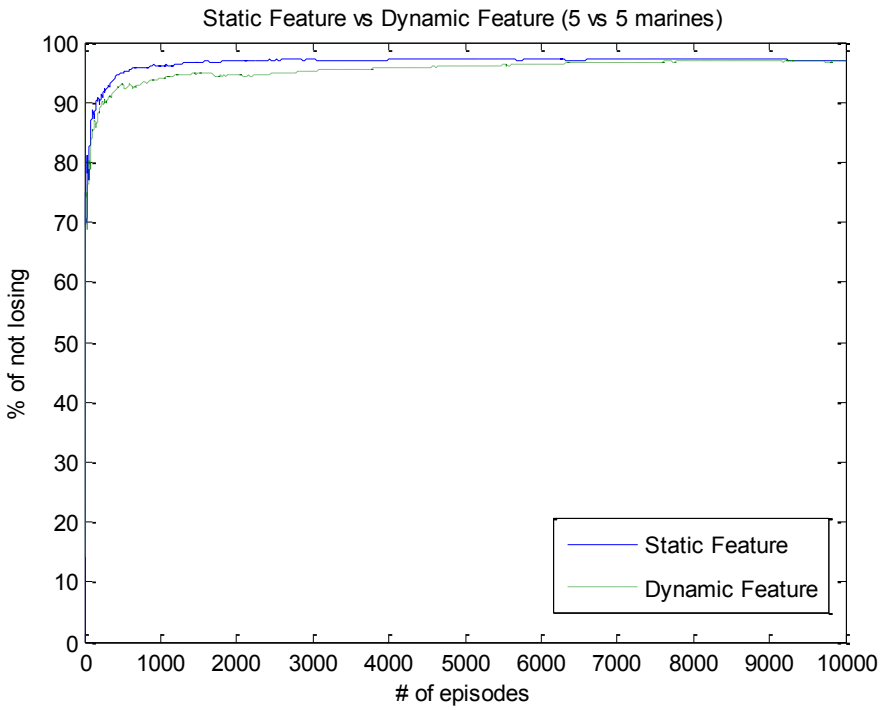

Figure IV-4 - Static vs. Dynamic Features (5 vs. 5 marines)

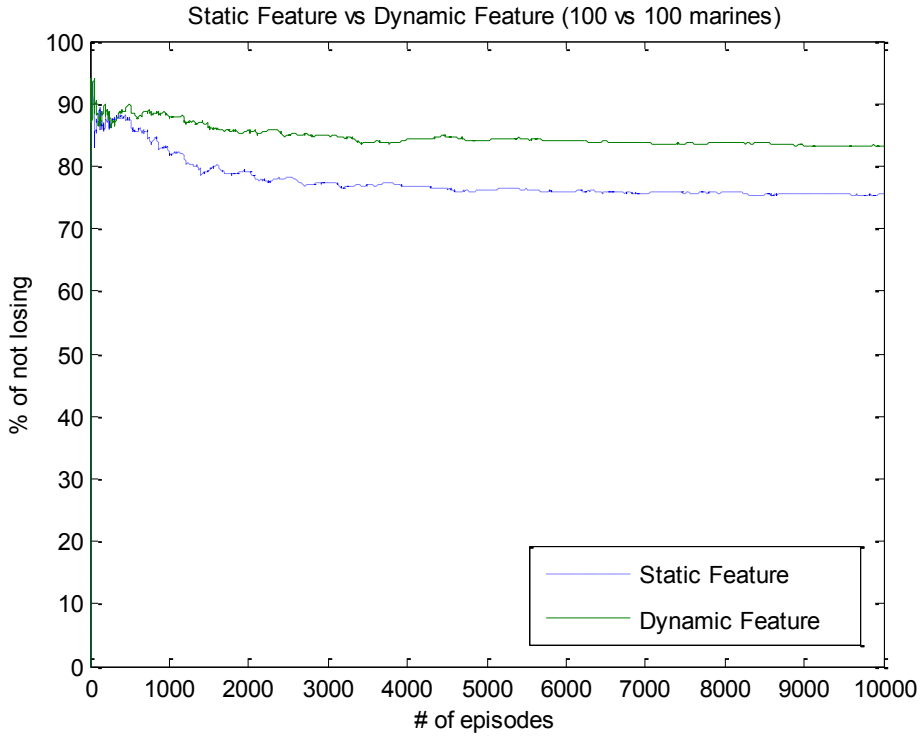

Figure IV-5 - Static vs. Dynamic Features (100 vs. 100 marines)

In Figure IV-6, Figure IV-7 and Figure IV-8, results of the clustering of ART2a network for the dynamic feature "number of enemies around" is shown. It is obvious that the increase in the number of enemies around from 20 to 100 does not increase the number of classes much, but rather the distribution of the values. This is an interesting result in the sense that, it is not counter intuitive to expect the number of classes to increase dramatically when the number of effectors are increased from 20 to 100 but these figures show the opposite. The adaptation to the new environment settings occur in basically in the clustering of the values, not the number of classes. This is an advantage because the complexity does not increase linearly with the number of effectors.

In the 5 vs 5 case Figure IV-6 - ART2a Classes (5 vs. 5), there are 5 clusters with values 0, 1, 2 and 5 are classified independently and values 3 and 4 are in class 4. In the 20 vs 20 case Figure IV-7 - ART2a Classes (20 vs. 20), the number of classes increases to 10. Values from 0 to 5 are classified in differently while (6:8) range, (9:13) range and (14:20) range are classified in groups. In the 100 vs 100 case Figure IV-8 - ART2a Classes (100 vs. 100), the number of classes increase to only 11, 0 to 5 are still classified in distinct classes. (6:7), (8:11), (12:20), (21:45) and (46:100) form other ranges for classes.

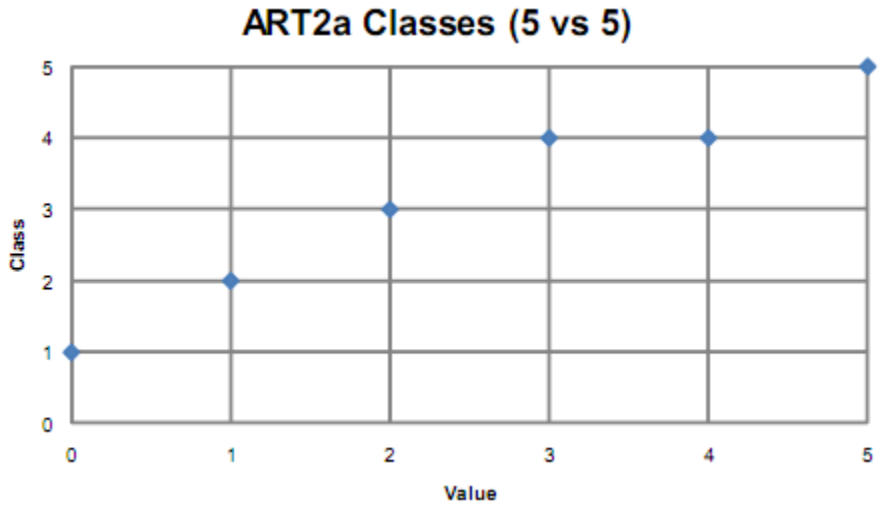

Figure IV-6 - ART2a Classes (5 vs. 5)

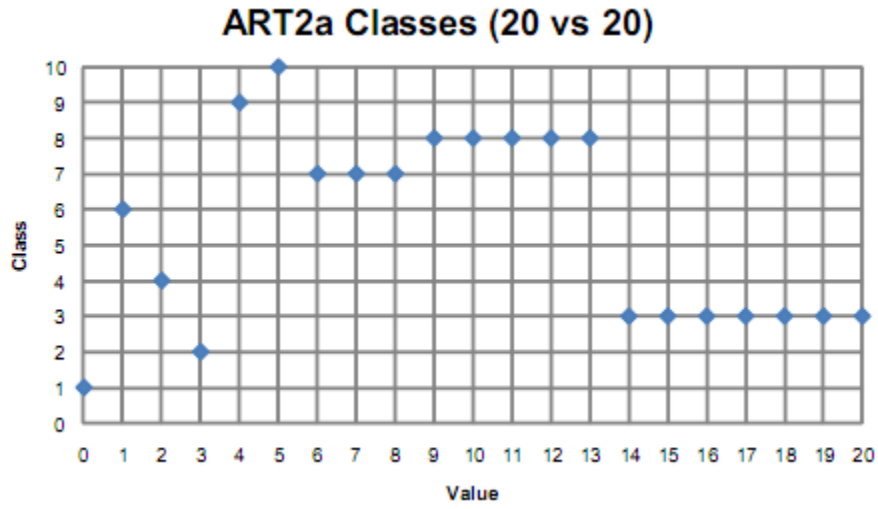

Figure IV-7 - ART2a Classes (20 vs. 20)

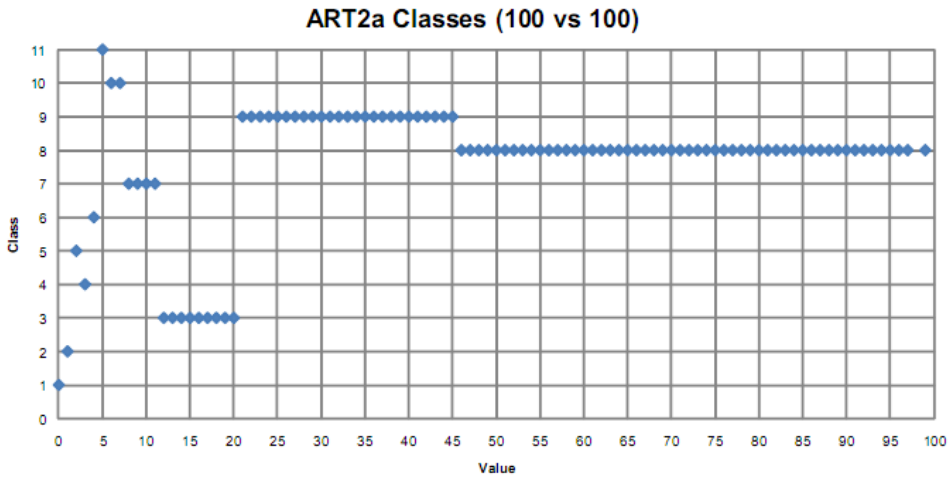

Figure IV-8 - ART2a Classes (100 vs. 100)

IV.4 Feature Selection and Dynamic Features

We have defined different features but not all of these features add the same value to the learning process, we have to compare and pick the best subset of these features. For that reason, we ran two experiments; first one is using one feature at a time. We did not specifically used methods in the literature such as Forward Feature Selection or Backward Feature Selection for a couple of reasons. Firstly, the number of defined features is limited to a small number. Secondly, the test for the performance of individual features showed that only some of the features are useful. Lastly, the increase

in the number of features led to divergence. The results in Figure IV-9, Figure IV-10, Figure IV-11 and Figure IV-12 show that the feature labeled Feature 1, which is the number of enemies around is the most useful, while other features add little value. This is surprising because common sense tells us that more detail will create a better model but as it is obvious in the second experiment with the combination of features in Figure IV-13, Figure IV-14 and Figure IV-15, adding features decreased the general performance. The reason for that drop in the performance is due to the linear nature of the linear function approximation, adding these features prevent the function to converge, this is related to the linear separability, a linear function can only separate data that is linearly aligned (Alpaydin, 2004). The reason for this divergence is fully studied in Tsitsiklis and Van Roy's work (Tsitsiklis & Van Roy, 1996). In Figure IV-10, Figure IV-11, Figure IV-9 and Figure IV-12, feature comparisons show that F1 is significantly more useful than other features. In these figures comparison of F1 with other features are shown. These figures also confirm that, addition of features other than F1 to the F1 feature itself in combination does not result in a better convergence and value in Figure IV-13 - F1 + F2 vs. F1 + F2 + F3 and Figure IV-14 - F1 + F2 + F3 vs. F1 + F2 + F3 + F4. Note that addition of F5 to the combination actually causes a divergence in Figure IV-15 - F1 + F2 + F3 + F4 vs. F1 + F2 + F3 + F4 + F5.

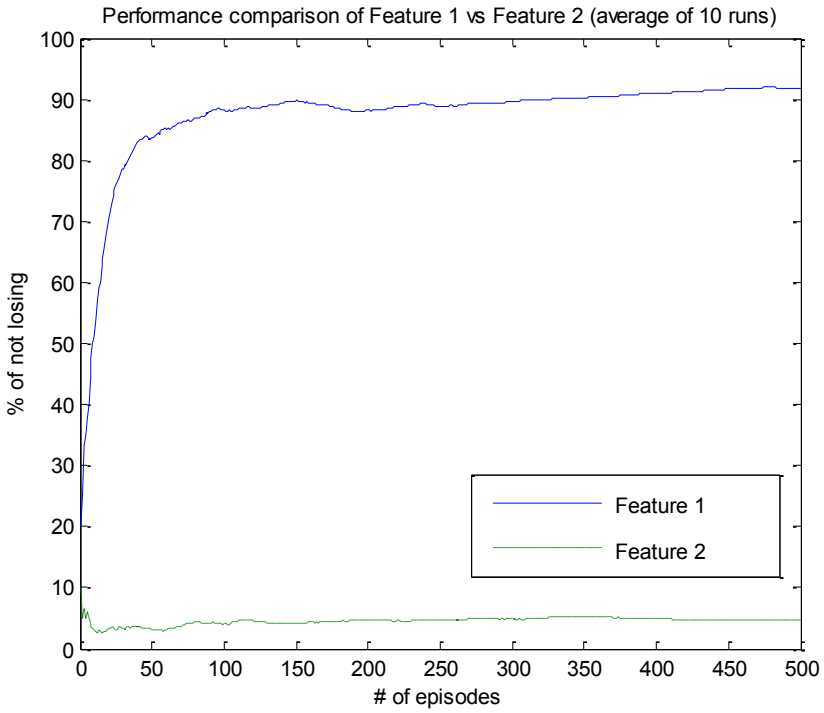

Figure IV-9 - F1 vs. F2

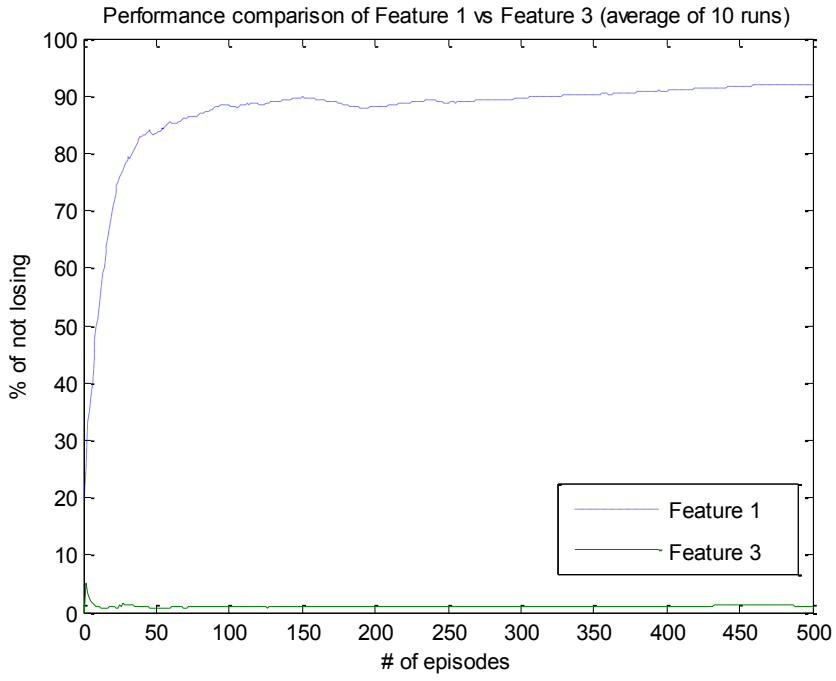

Figure IV-10 - F1 vs. F3

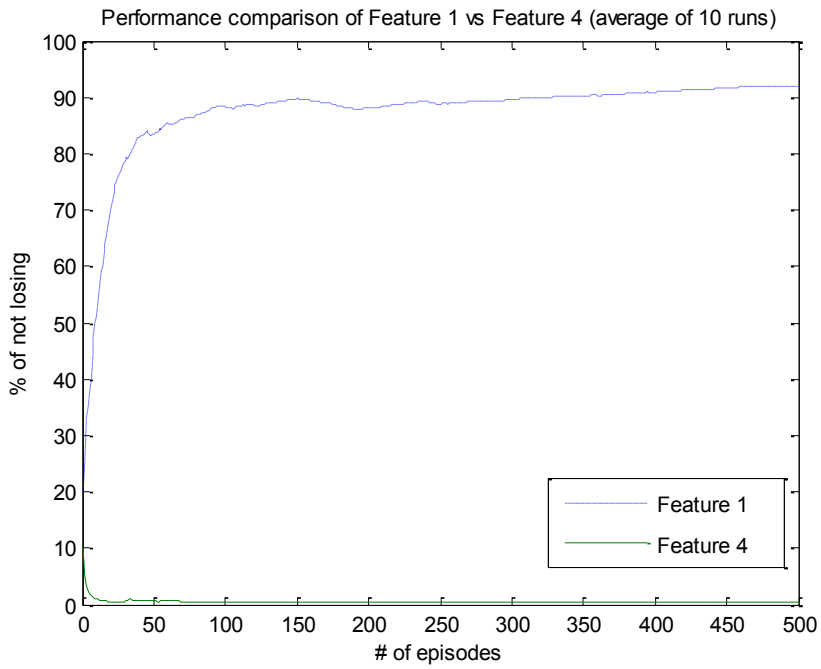

Figure IV-11 - F1 vs. F4

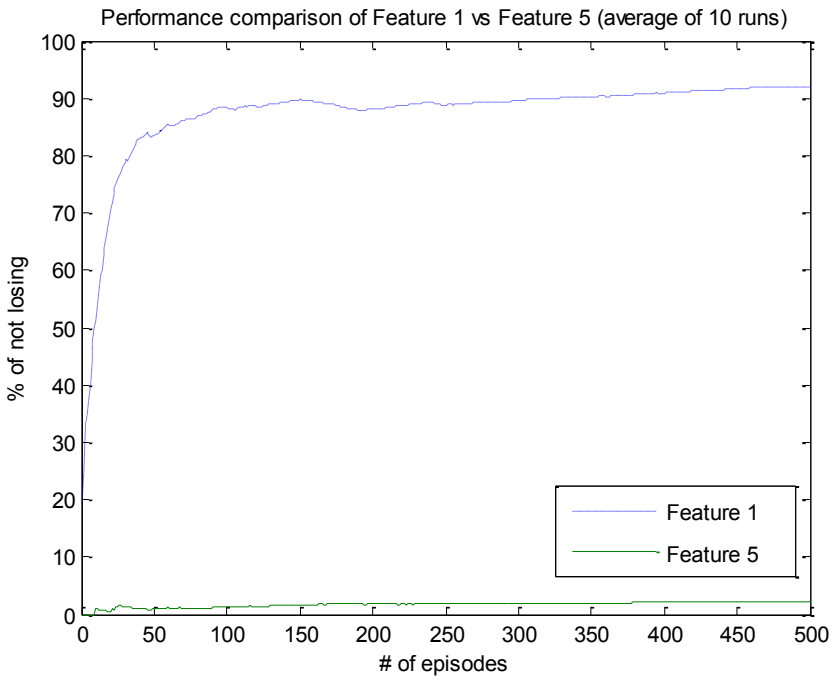

Figure IV-12 - F1 vs. F5

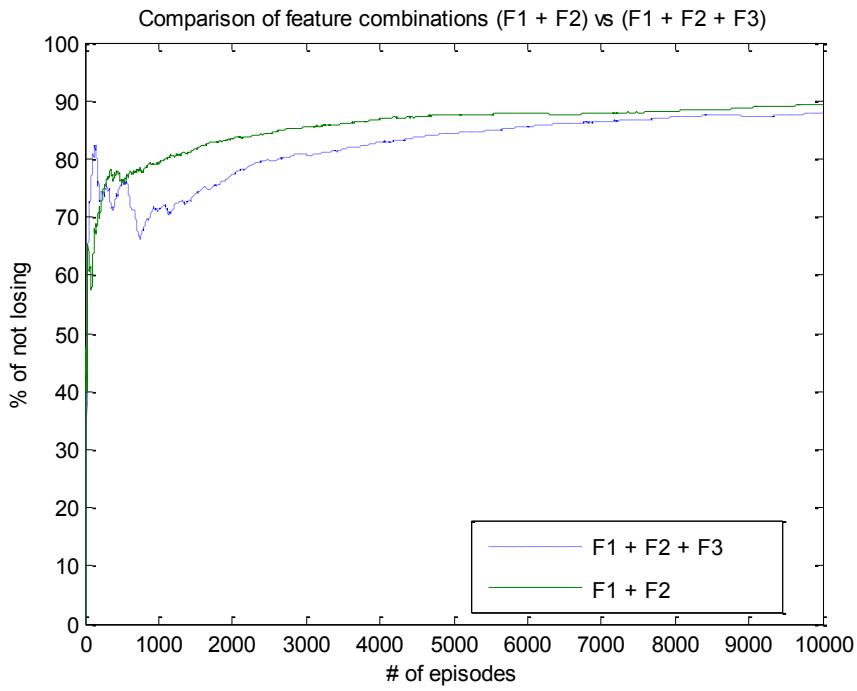

Figure IV-13 - F1 + F2 vs. F1 + F2 + F3

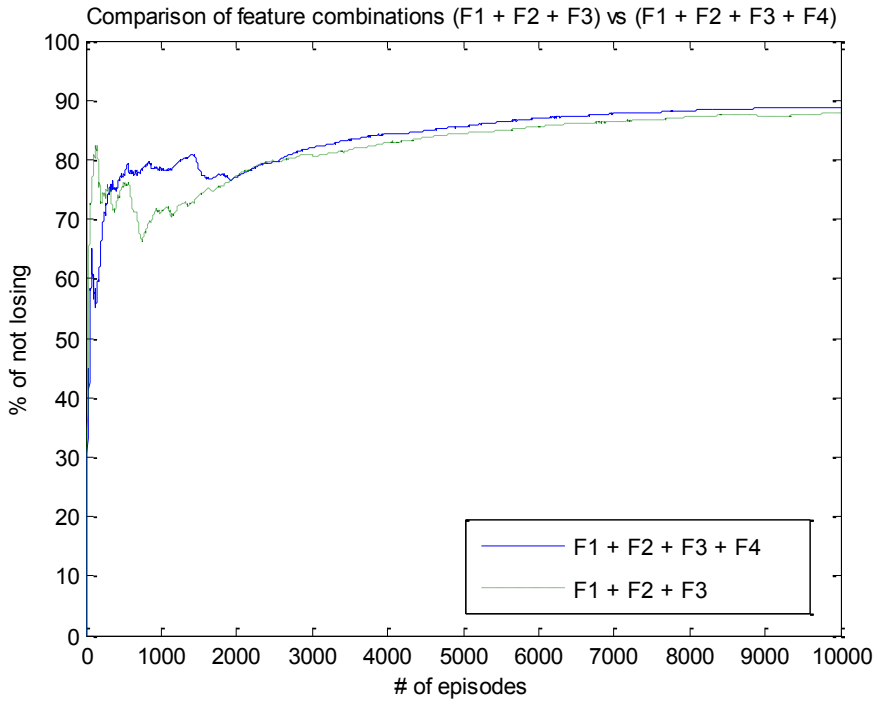

Figure IV-14 - F1 + F2 + F3 vs. F1 + F2 + F3 + F4

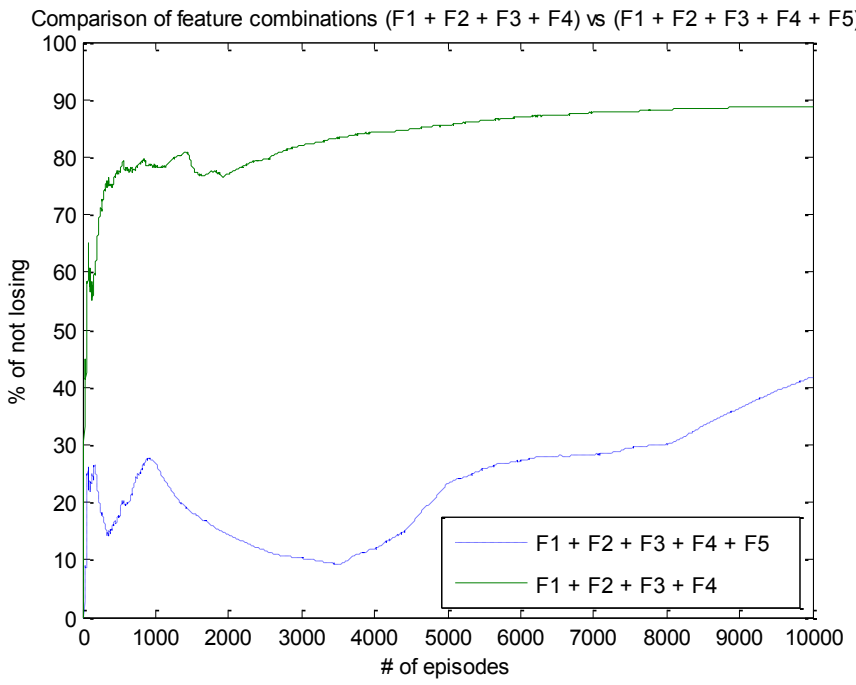

Figure IV-15 - F1 + F2 + F3 + F4 vs. F1 + F2 + F3 + F4 + F5

IV.5 RESOURCE PLANNING PROBLEM

In this last experiment, in addition to workers and marines, we add the base to the game and it becomes a fully featured RTS game with resource collection, unit training and sophisticated combat strategies. Base makes resource collection and unit training possible so this experiment is really about resource planning.

In the previous section, we conducted experiments with marines on strategic combat; this experiment includes these previous experiments while focusing on the base and its learning capability. The game map is a 20 by 20 grid, our agent starts on the position (1, 1) while enemy agent starts on the position (20, 20). They both start with one base and a single worker. Mineral patch is located in the middle at position (10, 10) so that they are equidistant to the main resource. None of the opponents has any mineral to start with, so to train a new unit, they should first collect enough resources. A worker or marine cost 10 minerals to train. There are 1000 minerals in the mineral patch, so the maximum number of units trained in one episode is limited to 100. These units may or

may not exist in the game at the same time. When all units of an opponent are destroyed or maximum number of time steps for an episode is reached, that is 1000 time steps, the episode terminates. The rewards are distributed at the end of the episode. The result of the game is not known until the episode ends, but to enable learning within the SMDP framework, especially for the marines to learn sophisticated combat strategies, a -1 reward is given per time step added by a +5 reward if the marine kills the attacked unit or a -5 penalty if the marine dies during an attack. This 5 reward or penalty mechanism is necessary for the marine to kill the enemy unit as soon as possible, and the -1 reward enables to do this quickly while also motivating the marine to do navigation options as quickly as possible. Enemy marines follow a fixed policy as in the strategic combat experiment, they attack what they see. Worker policy is picking up minerals from the mineral patch and returning to the base to drop them off. Enemy base follows a fixed policy of its own, training new workers if the number of workers is less than 5, otherwise it trains new marines.

In Figure IV-16, performances of the single agent and the multi-agent are compared as the percentage of not losing. Both of the sides start with a base unit and a worker unit. There is a mineral patch with 100 minerals. We can see that multi-agent outperforms the single agent case. Heterogeneous separation of agents leads to better specialization and faster convergence. In the multi-agent hierarchy, the agents do not have communication in between. They are not aware of each other existence and they don't need to, since they are specialized in different areas and they observe what they need to.

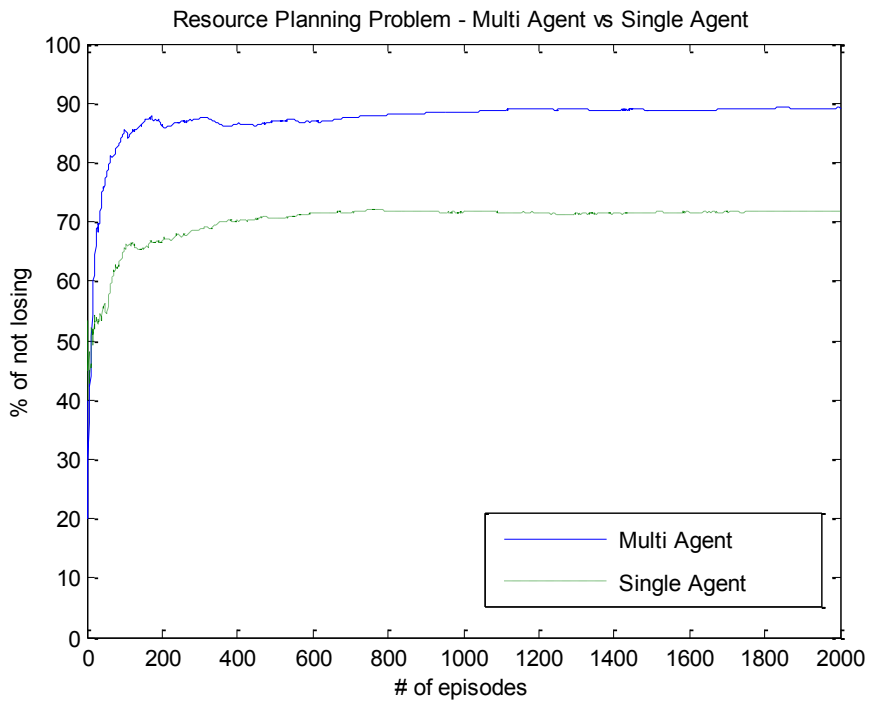

Figure IV-16 - Resource Planning - Multi Agent vs. Single Agent

As a result of this experiment, we can conclude that multi agent model outperformed single agent model in learning to beat an aggressive opponent in the game of RTS. It has learned to do resource planning in building an army that is both balanced in resource allocation and strategic combat. This last experiment showed that our agents designed in the previous experiments for resource collection and strategic combat with an added resource planning base unit in this experiment converged to a better policy when compared to a single agent model. Our multi agent model learned to beat the enemy 90% of the time while the single agent model managed to converge to only 70% winning ratio when not using the advanced techniques we applied in our work.

CHAPTER V.

CONCLUSION

RL framework is a broad subject with a lot of opportunities for designing and implementing learning agents in various different problem domains. We have applied RL techniques to a quite sophisticated problem and executed different experiments focusing on different aspects of the game, from delayed rewards to temporally abstract actions, from feature and option templates per unit to dynamic feature sets. We implemented a multi-agent abstraction, separated the marine units and base units in order to achieve a more efficient learning and better convergence. This leads to a better separation and better design at the cost of time and expertise needed to engineer those parameters and design.

In our first experiment, resource collection problem, we concluded that Linear Function Approximation with sound feature choices leads to a reduced state space and faster convergence, while additional features lead to even faster convergence. In the second experiment, strategic combat problem, we firstly used static features and compared the performance of static features in problems with different number of effectors. We concluded that static features only perform well for a specific range of number of effectors but when the number of effectors in the game, thus the state space, increases, they converge to a worse performing sub-optimal policy. Secondly we utilized a dynamic feature selection schema with ART2A to construct the dynamic features during the game play to overcome the limitations of static features. Usage of dynamic features lead to a better generalization and better convergence compared to static features when the number of effectors were increased. In the third experiment, resource planning problem, we showed that a multi agent model that uses a marine agent for strategic combat and a base agent for resource planning by differentiating agents in a multi agent structure lead to a better convergence compared to a single agent that

attempts at learning the same task. Multi agent model outperformed the single agent model by 20% in winning ratios against the enemy agent.

There are many points to improve for possible future work. It might be possible to introduce more clever features to create a better and more compact representation of the real world state. Another improvement might be, automatic feature extraction from the environment with a given initial set of basic features, then as the learning proceeds features that have lower impact on the function approximation can be discarded.

BIBLIOGRAPHY

Alpaydin, E., 2004. *Introduction to Machine Learning*. MIT Press.

Bower, E.R. & Gordon, H., 1975. *Theories of Learning, Englewood Cliffs, NJ, fourth edition*. Prentice-Hall.

Bradtke, S.J. & Duff, M.O., 1994. Reinforcement learning methods for continuous-time Markov decision problems. In *Advances in Neural Information Processing Systems*., 1994. MIT Press.

Carpenter, G.A., Grossberg, S. & Rosen, D.B., 1991. ART 2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition. *Neural Networks*.

Dietterich, T.G., 2000. Hierarchical reinforcement learning with the MAXQ value function. *Journal of Artificial Intelligence Research*, pp.227-303.

Dietterich, T.G., 2008. Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the 25th international conference on Machine learning*. Helsinki, 2008.

Eugene, A.F. & Shwartz, A., 2001. *Handbook of Markov Decision Processes*. Springer.

Ghavamzadeh, M. & Mahadevan, S., 2007. Hierarchical Average Reward Reinforcement Learning. *Journal of Machine Learning Research*.

Guestrin, C. & Gearhart, C., 2003. Generalizing plans to new environments in relational MDPs. *IJCAI*.

Hsu, F.-h., 2002. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press.

Kaelbing, P.L., Littman, L.M. & Moore, A.W., 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, pp.237-85.

Lee, J., Koo, B. & Oh, K., 2008. State space optimization using plan recognition and reinforcement learning on RTS game., 2008.

Madeira, C., Corruble, V., Ramalho, G. & Ratitch, B., 2004. Bootstrapping the Learning Process for the Semi-automated Design of a Challenging Game AI. In *n Proceedings of the AAAI Workshop on Challenges in Game AI*. San Jose, CA, 2004.

Parr, R., 1998. *Hierarchical control and learning for Markov decision processes*. University of California, Berkeley.

Ponsen, M.J.V., Muñoz-Avila, H., Spronck, P. & Aha, D.W., 2006. Automatically Generating Game Tactics via Evolutionary Learning. *AI Magazine*, pp.75-84.

Ponsen, M., Spronck, P. & Tuyls, K., 2006. Hierarchical Reinforcement Learning with Deictic Representation in a Computer Game. In *18th Benelux Conference on Artificial Intelligence*., 2006.

Russell, S. & Norvig, P., 2003. *Artificial Intelligence A Modern Approach*. Prentice Hall.

Russel, S., Marthi, B. & Latham, D., 2005. Concurrent Hierarchical Reinforcement Learning. In *In Proceedings IJCAI*., 2005.

Russel, S. & Parr, R., 1998. *Reinforcement learning with hierarchies of machines*. MIT Press.

Sharma, M. et al., 2007. Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. *IJCAI*, pp.1041-46.

Spronck, P., Ponsen, M., Sprinkhuizen-Kuyper, I. & Postma, E., 2006. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3), pp.217-48.

Sutton, R. & Barto, A.G., 1998. *Reinforcement Learning*. MIT Press.

Sutton, R., Precup, D. & Singh, S., 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, pp.112:181-211.

Tesauro, G., 1995. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*.

Tsitsiklis, J.N. & Van Roy, B., 1996. Feature-Based Methods for Large Scale Dynamic Programming. *Machine Learning 22*, pp.59-94.

Tsitsiklis, J.N. & Van Roy, B., 1997. An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control*.

Williams, R.J., 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8, pp.229-56.

Ying-zi, W. & Ming-yang, Z., 2003. Effective strategies for complex skill real-time learning using reinforcement learning. In *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing*., 2003.

CURRICULUM VITAE

M E H M E T C İ H A N K U R T

Ataköy 4. Kısım O-188 2/4 34158 Bakırköy / İSTANBUL

Phone : +90 212 560 24 27/ GSM: +90 533 641 24 08

e-mail: cihan.kt@gmail.com

EDUCATION

2006 – … : Marmara University (MS in Computer Engineering), ISTANBUL
1997 – 2005 : Boğaziçi University (BS in Computer Engineering), ISTANBUL
1990 – 1997 : Ar-el College

WORK EXPERIENCE

Inveon Information Systems, Istanbul – Software Engineering Manager (2006 Oct - …)
Altaca Group, Istanbul – Software Engineer (2003 Feb - 2005 Dec)
Loquisoft, Vienna – Software Engineer (2006 Jan – 2006 Jun)
OYAK Renault, Bursa –Sofware Development Specialist (2002 Sep – 2003 Sep)

PERSONAL INFORMATION

Birth Date and Place : 07/05/1979 - İstanbul

RESEARCH INTERESTS

Machine Learning, Reinforcement Learning.