Massey University

# Action-Selection in RoboCup Keepaway Soccer: Experimenting with Player Confidence

A thesis presented in partial fulfilment of the requirements for the degree of

Masters of Science

In

Computer Science

at Massey University, Albany,

New Zealand.

Samara Ann Neilson

2006

## Abstract

Through the investigation of collaborative multi-agent domains, in particular those of robot soccer and robot rescue, and the examination of many popular action-selection methodologies, this study identifies some of the issues surrounding entropy, action-selection and performance analysis.

In order to address these issues, a meaningful method of on-field player evaluation, the confidence model, was first proposed then implemented as an action-selection policy. This model represented player skill through the use of percentages signifying relative strength and weakness and was implemented using a combination of ideas taken from Bayesian Theory, Neural Networks, Reinforcement Learning, Q-Learning and Potential Fields.

Through the course of this study, the proposed confidence model action-selection methodology was thoroughly tested using the Keepaway Soccer Framework developed by Stone, Kuhlmann, Taylor and Liu and compared with the performance of its peers.

Empirical test results were also presented, demonstrating both the viability and flexibility of this approach as a sound, homogeneous solution, for a team wishing to implement a quickly trainable performance analysis solution.

## Acknowledgements

I would like to gratefully acknowledge the enthusiastic supervision of Dr. Chris Messom during this work. I would also like to thank Stephen Ford for the assistance provided dealing with Helix's technical problems. And, finally, I am forever indebted to my friends and family for their understanding, endless patience, encouragement and support when it was most required.

## Table of Contents


Abstract i

Acknowledgements i

List of Tables v

List of Figures vi

**Chapter 1: Introduction & Study Background**

1.1 Introduction 1

1.2 Statement of the Problem 4

1.3 General Research Hypotheses 4

1) Collaborative Multi-Agent Domains 5

2) Entropy and Action-Selection 5

3) Assessing Individual Performance 6

4) Implementation 6

5) Performance Assessment 7

1.4 Significance of the Study 7

1.5 Definitions & Operational Terms 7

1.6 Delimitations 9

**Chapter 2: Collaborative Multi-Agent Domains**

2.1 Introduction 10

2.2 Robot Soccer 16

2.2.1 Robot Soccer Initiatives 17

2.2.1.1 RoboCup Soccer 17

2.2.1.2 FIRA 19

2.2.2 Robot Soccer Simulators 21

2.2.2.1 RoboCup Soccer Server 22

2.2.2.2 SimuroSot 3D Soccer Simulator 23

2.2.2.3 TeamBots 25

2.2.2.4 Webots 26

2.2.3 Real Robot Soccer 27

2.2.4 Soccer Keepaway 30

2.3 Robot Search and Rescue 31

2.3.1 Robot Rescue Initiatives 32

2.3.1.1 RoboCup Rescue 32

2.3.1.2 NIST 33

2.3.2 Robot Rescue Simulation 34
2.3.2.1 RoboCup Rescue Simulator 34
2.3.3 Real Robot Rescue 35
2.4 Summary 36

**Chapter 3: Action Selection Methodologies and Implementations**

3.1 Action Selection Methodologies 38
3.1.1 Bayesian Theory 38
3.1.2 Neural Networks 39
3.1.3 Potential Field Methodologies 41
3.1.4 Reinforcement Learning 42
3.1.5 Q-Learning 44
3.2 Implementations
3.2.1 Simulation Environment 45
3.2.1.1 RoboCup Soccer Server 45
3.2.1.2 Keepaway Soccer Framework 47
3.2.2 Action-Selection Implementations 50
3.2.2.1 Confidence Model 51
3.2.2.2 Direct Implementation 52
3.2.2.3 Potential Field Confidence Model Implementation 54
3.3 Empirical Testing Regime 55
3.3.1 Benchmark Testing 55
3.3.2 Feasibility Testing 56
3.3.3 Scalability Testing 56
3.3.4 Entropy Testing 56
3.4 Summary 57

**Chapter 4: Results**

4.1 Benchmark Testing 59
4.2 Feasibility Testing 60
4.2.1 2 vs. 1 Keepaway 61
4.2.2 3 vs. 2 Keepaway 64
4.2.3 4 vs. 3 Keepaway 67
4.2.4 5 vs. 4 Keepaway 71
4.2.5 Summary 74
4.3 Scalability Testing 75
4.3.1 3 vs. 2 Keepaway on Differing Field Sizes 76

4.3.1.1 15 Metre Field 76
4.3.1.2 40 Metre Field 79
4.3.2 11 vs. 11 Keepaway 82
4.3.3 Summary 85
4.4 Entropy Testing 87

**Chapter 5: Discussions and Conclusion**
5.1 Discussion 90
5.1.1 Keepaway Soccer Framework 90
5.1.2 Confidence Model Observations & Results 91
5.2 Conclusion 94
5.3 Future Work 98

**Chapter 6: References** 99

**Appendices:**
Appendix A1: User Guide & Code Printout 104
A1.1 User Guide 104
A1.1.1 Installation 104
A1.1.2 Configuration Settings 104
A1.1.3 Sample Startup Script 105
A1.2 Printout of `LearningAgent.h` Code 106
A1.3 Printout of `LearningAgent.cc` Code 108
Appendix A2: Result Graphs 119
A2.1 Feasibility Testing Graphs 120
A2.2 Scalability Testing Graphs 132
A2.3 Entropy Testing Graphs 141

## List of Tables


| | | |
|---|---|---|
| Table 2.1: | Differences between the RoboCup Rescue Simulator and the RoboCup Soccer Server. | 35 |
| Table 3.1: | Feasibility testing regime. | 56 |
| Table 4.1: | Keepaway framework results provided by Stone et al (2005). | 59 |
| Table 4.2: | Results recorded from testing conducted over 10,000 episodes on the Helix computer system. | 59 |
| Table 4.3: | Observed difference between results benchmarked by Stone et al and collected from Helix. | 60 |
| Table 4.4: | Summary of results for 2 vs. 1 keepaway soccer. | 63 |
| Table 4.5: | Summary of results for 3 vs. 2 keepaway soccer. | 66 |
| Table 4.6: | Summary of results for 4 vs. 3 keepaway soccer. | 70 |
| Table 4.7: | Summary of results for 5 vs. 4 keepaway soccer. | 73 |
| Table 4.8: | Policy performance summary for results collected using Helix. | 74 |
| Table 4.9: | Summary of results for 3 vs. 2 keepaway soccer on a 15m x 15m field. | 78 |
| Table 4.10: | Summary of results for 3 vs. 2 keepaway soccer on a 40m x 40m field. | 81 |
| Table 4.11: | Summary of results for 11 vs. 11 keepaway soccer. | 84 |
| Table 4.12 | Policy performance results collected during 3 vs. 2 feasibility testing and 3 vs.2 scalability testing. | 85 |
| Table 4.13: | Performance results collected during feasibility testing and 11 vs. 11 scalability testing. | 86 |
| Table 5.1: | Average performance comparison between version 0.4 and 0.6 of the Framework. | 93 |
| Table 5.2: | Performance comparison between the 'Potential Fields' policy and 'Reinforcement Learning' policy for 3 vs. 2 keepaway over differing field sizes. | 94 |
| Table A1.1: | Keepaway settings. | 104 |

## List of Figures


| | | |
|---|---|---|
| Figure 2.1: | A taxonomy of autonomous agents. | 15 |
| Figure 2.2: | FIRA logo and examples of participant classes. | 19 |
| Figure 2.3: | RoboCup Soccer Server. | 22 |
| Figure 2.4: | Screenshot of the SimuroSot Simulator in action. | 24 |
| Figure 2.5: | Screenshot of SoccerBots showing set kick-off positioning. | 25 |
| Figure 2.6: | Simulated soccer game between two Sony AIBO robots. | 26 |
| Figure 2.7: | Small-sized team at RoboCup 2005. | 28 |
| Figure 2.8: | Medium-sized team at RoboCup 2005. | 28 |
| Figure 2.9: | Four-legged team at RoboCup 2005. | 29 |
| Figure 2.10: | Humanoid robot from team Osaka getting ready to compete at RoboCup 2005. | 29 |
| Figure 2.11: | Example of an episode with 3 keepers and 2 takers playing in a 20m x 20m region. | 30 |
| Figure 2.12: | RoboCup Rescue simulator in action at RoboCup 2005. | 34 |
| Figure 3.1: | Artificial neuron with input vectors (x), weighted input lines (w) and a thresholding function f()that determines the neurons output value. | 40 |
| Figure 3.2: | Example network topologies for the PassEvaluate() and GetOpen() functions. | 41 |
| Figure 3.3: | Uniform, attractive, repulsive and tangential potential fields. | 42 |
| Figure 3.4: | Model of a player's visual sensor showing its view cone and visible distance where any agents seen can be identified with a confidence of 100%. This confidence decreases by given proportions as players move farther outside the visible distance into the marked zones. | 47 |
| Figure 3.5: | Starting positions for 3 vs. 2 keepaway soccer. | 48 |
| Figure 3.6: | Representations of keepaway state variables. | 49 |
| Figure 3.7: | Human soccer player decision scenario. | 50 |
| Figure 3.8: | A direct confidence level approach to action-selection. | 53 |
| Figure 3.9: | A non-optimal passing scenario is displayed to the left with a visualisation of the beginning stages of the potential fields hybrid solution. The results of the solution, once optimised, are shown to the right along with the emergent behaviour. | 55 |

| | | |
|---|---|---|
| Figure 4.1: | Policy Performance Graph for 2 vs. 1 keepway. | 61 |
| Figure 4.2: | Box and whisker analysis of 2 vs. 1 policy performance. | 62 |
| Figure 4.3: | Graph depicting ratio of episode outcomes for 2 vs. 1 keepaway. | 63 |
| Figure 4.4: | Policy performances for 3 vs. 2 keepaway. | 64 |
| Figure 4.5: | Box and whisker analysis of 3 vs. 2 policy performance. | 65 |
| Figure 4.6: | Graph depicting the ratio of outcomes of 3 vs. 2 keepaway. | 66 |
| Figure 4.7: | Policy performance graph for 4 vs. 3 keepaway. | 68 |
| Figure 4.8: | Box and whisker analysis of 4 vs. 3 keepaway soccer performance. | 69 |
| Figure 4.9: | Outcome ratios for 4 vs. 3 policy performance. | 70 |
| Figure 4:10: | Policy performance graph for 5 vs. 4 keepaway. | 71 |
| Figure 4.11: | Box and whisker analysis of 5 vs. 4 keepaway soccer performance. | 72 |
| Figure 4:12: | Outcome ratios for 5 vs. 4 policy performance. | 73 |
| Figure 4:13: | Policy performance summary for results collected using Helix. | 75 |
| Figure 4:14: | Policy performance graph for 3 vs.2 keepaway played on a 15m x 15m field. | 76 |
| Figure 4.15: | Box and whisker analysis of 3 vs.2 keepaway performance on a 15m x 15m field. | 77 |
| Figure 4:16: | Outcome ratios for 3 vs. 2 policy performance on a 15m x 15m field. | 78 |
| Figure 4:17 | Policy performance graph for 3 vs.2 keepaway played on a 40m x 40m field. | 79 |
| Figure 4:18: | Box and whisker analysis of 3 vs.2 keepaway performance on a 15m x 15m field. | 80 |
| Figure 4:19: | Outcome ratios for 3 vs. 2 policy performance on a 40m x 40m field | 81 |
| Figure 4:20: | Policy performance graph for 11 vs.11 keepaway. | 82 |
| Figure 4:21: | Box and whisker analysis of 11 vs. 11 keepaway performance. | 83 |
| Figure 4:22 | Outcome ratios for 11 vs. 11 policy performance. | 84 |
| Figure 4:23 | 3 vs. 2 policy performance comparison for differing field sizes. | 86 |
| Figure 4:24: | Policy performance comparison for differing team configurations. | 87 |
| Figure 4:25: | Keeper passing confidences observed for a handicapped 3 vs. 2 team. | 88 |
| Figure 4:26: | Keeper passing confidences observed for a 3 vs. 2 team. | 89 |
| Figure 5.1: | "Piggy in the middle" passing exhibited by players using the Potential Fields policy implementation. | 92 |

# 1. Introduction & Study Background

## 1.1 Introduction

Early multi-agent collaboration primarily involved numbers of homogenous robots completing simple tasks such as foraging, navigating and path-finding with more recent research involving teams competing in robot soccer and teams of robots participating in rescue operations.

These latter collaborative multi-agent domains are considered to be both highly dynamic and highly complex with agents only able to sense partial and imperfect information from the world around them and complete actions whose effects are variable and may not be ascertainable for many cycles.

As such, analysis through brute-force comparison methods using broad evaluation statistics such as 'the number of goals scored', 'the number of fires put out', or 'the number of victims rescued' etc, proved insufficient as these evaluation criteria were easily influenced by the chaotic and opaque nature within which these performances were executed.

As robot rescue is a relative new-comer to the competition scene, any performance analysis takes the form of a direct strength and weakness comparison such as the analysis presented by Kleiner, Brenner and Brauer at the RoboCup competition in 2005. In their presentation they isolated a set of statistics and compared and contrasted their team's performance with other teams in order to gain a better understanding of the relative strengths and weaknesses of various simulated rescue approaches (Kleiner et al, 2005).

On the other hand, early analysis work in RoboCup concentrated heavily on the offline statistical analysis of game logs. This work was pioneered by Takahashi and Naruse (Takahashi et al, 1998) who began measuring the performance of teams and developing a set of statistical features for teams participating in RoboCup 1997.

Takahashi (Takahashi, 1999) continued this work and while finding no correlation between scoring and agent collaboration, he did note that teams more inclined to collaborative actions such as 1-2 passes, where player A passes to player B to avoid defender C as player B returns ball to player A, played more effectively than teams disinclined to such collaborative plays.

Tanaka-Ishii and (Tanaka-Ishii et al, 2000) completed a more thorough detailed statistical analysis of simulation teams from the 1997 and 1998 RoboCup competitions based on this early work by

Takahashi and Naruse. Their evaluation was two-fold, firstly completing an offline analysis using 32 evaluation features and secondly investigating the robustness of the teams by replaying games with reduced team sizes. From their findings, they were able to conclude that poorly performing teams were not necessarily all bad, just let down by one or more aspects of play. They also noted that many of these teams performed better with fewer players.

The move towards complicated tactics development that ensued, again saw many teams handicapping themselves, most notably documented in the 2000 competition where the CMUnited99 team, championship winners of the previous year, were re-entered without change and achieved a fourth placing although the team's code had been publicly available for a year (Stone et al, 2001). It could be argued that only two teams bettered the CMUnited99 team as the third was ATT-CMUnited2000, an updated version of its 1999 predecessor.

To date, most of the analysis tools available have been offline and only analyse generalities of the game such as number of passes, pitch ownership, player distribution etc. Most of this analysis work has since been put to use analysing opponent behaviour in order to predict future behavioural patterns, a move primarily driven by teams moving from more traditional action-selection methods to complicated learning algorithms which required large time investments in order to sufficiently train agents to learn basic and advanced skills.

Many ideas have since been tabled to improve this training time including a move to using smaller, less complex sub-domains such as the keepaway sub-domain to train teams (Stone et al, 2005a), and using online and offline coaches and trainers to hone players performances, which until that time were most commonly seen in professional sports (between humans).

With this move towards greater complexity in team game play and tactics came a push for the development of more fully heterogeneous agents. This push could also be associated with the introduction of robot rescue competitions in the early part of the 2000s which almost exclusively called for the use of heterogeneous agents and the subsequent inclusion of heterogeneous agents in version 7 of the RoboCup Soccer Server in 2002.

Teams participating in robot rescue and robot soccer often carry the blanket description of "heterogeneous" seemingly due to the perception that the inherent complexity of these domains would be best handled by heterogeneous teams. But are these teams actually heterogeneous?

Although many methods of calculating entropy, the level with which a team can be described as homogeneous or heterogeneous, exist, there seems to be much confusion as to how these equations should be applied so a true accounting of entropy can be ascertained.

This confusion seems to be drawn from two opposing arguments, observed during the review of literature surrounding collaborative multi-agent domains, and noted especially in domains involving software agents or where software was the sole differentiating factor.

For teams which rely on hardware diversity for advantage, evident in the robot rescue domain and to a smaller extent in the medium and large-sized robot soccer leagues, heterogeneous entropy can be easily ascertained by inputting the number of points of difference in the robots physical design.

But what about the entropy of the robots software components? For teams which rely solely on software for differentiation, such as teams participating in the simulated, four-legged and small-sized leagues of RoboCup, entropy isn't so clear cut as the agents within a team could be using identical hardware platforms and running identical code.

Up to the introduction of dynamic role assignment proposed by Stone and Veleso (Stone & Veleso, 1999), which enabled teams of identically coded agents to switch roles on-the-fly from attacker to attack support, and from attack support to defence for example, based on the perceived game state at that point of time, the requirements of determining entropy became further blurred.

To add to this confusion, more often than not, the designers of these teams take the more literal, bi-polar approach and state that their teams are homogeneous, as they shared identical code, and depending on the application, identical hardware, but another school of thought soon emerged, which was most notably documented by Blach (Balch, 2000) and utilised in works by both Gerkey and Mataric, (Gerkey & Mataric, 2004) and Prokopenko & Wang (Prokopenko & Wang, 2004).

Balch theorised that any degree of behavioural difference needed to be accounted for, so using this argument, teams utilising dynamic role assignment, or any similar method which would see agents implementing different behaviours during the same game scenario at the same time, should be considered as a heterogeneous team, to a certain degree (Balch, 2000), effectively removing the bi-polar scale in favour of one made up of shades of grey.

With these two opposing viewpoints in mind, there seems to be a general confusion over the role heterogeneity plays in the software components of agents. If the designer(s) of what they consider to be a homogeneous robot team believe that their opponents will be acting in an equally

homogeneous fashion, does it influence their own choice of action-selection policies for their team? Do any of the popular action-selection policies take this diversity into account?

To take an example from the other popular collaborative multi-agent domain of simulated robot rescue: a rescue team as a whole is made up of say three teams of agents, police agents, ambulance agents and fire-fighting agents, each type with it's own strengths and weaknesses. An observer would be clearly justified, under both interpretations of entropy, in stating that this team is heterogeneous, and indeed, this has become the general perception, that all rescue tasks can only be completed by heterogeneous teams.

But what if the individual teams are homogeneous in that they both share identical code and each agent would execute the same behaviour in the same scenario? Wouldn't a more accurate perception be that a combination of homogeneous and heterogeneous teams is superior for rescue applications rather than the current perception of heterogeneous team superiority?

These issues surrounding multi-agent collaboration, action-selection and entropy are central to this study and will be presented and discussed in further detail in the sections and chapters which follow.

## 1.2 Statement of the Problem

From an investigation of collaborative multi-agent domains, in particular the domains of robot soccer and robot rescue, this study will select one domain which will become the test bed for an examination of the role entropy plays within that domain and its effect, if any, on action-selection.

This study will then propose a method of action-selection which allows the performance of individual agents to be empirically analysed. This method will then be compared with its peers in order to accurately gauge its performance and suitability.

## 1.3 General Research Hypotheses

The general research hypotheses which provide a framework for this study are based upon observations of multi-agent collaborative domains described above and in the subsequent reviews of literature.

The areas which will be explored in this study are: 1) Collaborative multi-agent domains, with emphasis on the robot soccer and robot rescue domains; 2) The idea of the perceived entropy of

teams impacting the choice of action-selection methodologies used; 3) Developing a method of analysis which can by used to assess individual agents within a team based on their individual performances; 4) Incorporating this analysis and assessment method into a control policy for a team of agents; and 5) Analysing the performance of such a team. These areas are discussed in more detail below.

1) Collaborative Multi-Agent Domains

The first part of this discussion concerns the question of what is the definition of a collaborative multi-agent and what are the domains that they are used in. Thus, the first research question asked in this study is:

1. What is involved in the two large collaborative multi-agent domains of robot soccer and robot rescue? And which domain is better suited to conduct development and experimentation in?

Chapter 2 of this study addresses this research question by looking at the varying definitions of an agent and what makes an agent collaborative and what level of cooperation must be achieved amongst agents before they are classified as a multi-agent group. Once these definitions have been refined, chapter 2 continues to illustrate work being conducted in these two main domains of collaborative multi-agent research: robot soccer and robot rescue. Finally this chapter concludes with a summary of the findings of this review of literature and discusses the conclusions drawn regarding which framework would be best suited to continue this investigation in.

2) Entropy and Action-Selection

The second general hypothesis concerns the question of how entropy and action-selection are related. Thus, the second and third research questions posed are:

2. What is action-selection and what are the popular methods utilised in the chosen research domain?

3. Are considerations of team entropy noticeably included or excluded in the workings of these action-selection policies?

Sections 3.1 and 3.2.2 of chapter 3 address these questions firstly, by defining action-selection and discussing popular methods and variations of these methods used today. Secondly, the entropy of

these methods is addressed and the underlying trend observed discussed, leading up to the next general hypothesis discussed below.

3) Assessing Individual Performance

The next part of this study concerns the question of whether an individual agent's abilities can be assessed and used in order to determine the next suitable course of action. This leads to the fourth research question:

4. How can an individual agent's strengths and weaknesses be assessed so that they can be compared and contrasted with its peers?

Section 3.2.2 of chapter 3 addresses this question firstly, by discussing the need for such an approach and secondly, by examining aspects of the popular analysis methods which can be combined to form what has been called the confidence model approach.

4) Implementation

The fourth research hypothesis examines how the above method of performance assessment can be implemented on a team of collaborative agents operating in the chosen domain and how this implementation can be described in terms of entropy. This is addressed by the fifth and sixth research questions:

5. How can this method of assessing relative strength and weakness be implemented as an action-selection policy in the chosen domain?

6. What is the actual entropy of a team using the above action-selection policies when it can be classed as both fully heterogeneous and fully homogeneous under both the bi-polar and social scales of entropy?

Section 3.2 of chapter 3 addresses the implementation of the proposed performance analysis method and proposes solutions to issues observed early in implementation which lead to the development of two separate implementations, one a direct implementation and one a hybrid method developed to improve the short comings of the first.

5) Performance Assessment

The final part of this study involves comparing and contrasting the performance of the developed assessment method with other available action-selection policies. The final research questions therefore address this testing phase as follows:

7. How does this method perform in comparison with other action-selection policies?

8. How does this method scale in comparison with other action-selection policies?

A testing plan is initially presented in the section 3.3 of chapter 3 with specific details of how tests were conducted with a presentation of the results collected found in chapter 4.

These results and their impact on the study are then discussed in chapter 5.

## 1.4 Significance of the Study

Through the proposed research hypotheses stated above, this study aims to explore some of the issues surrounding multi-agent domains, entropy, action-selection and performance analysis and helps define the use of confidence levels as a meaningful method of individual player assessment.

The study of aspects which influence multi-agent collaborative domains, in particular the fields of robot soccer and robot rescue are important as both have the potential to have significant impact on the quality of human life in the future. As both of these research fields are relatively new and both still many years away from the expected achievement of their 50 year goals which include aiming "to build a team of robot soccer players, which can beat a human World Cup champion team", the implication is that there is still much left to be examined and explored.

It is envisaged that this study will be useful to those interested in deepening their understanding of these applications of collaborative multi-agent theory particularly with the view towards the studies of entropy, action-selection and performance analysis. It is also envisaged that this study will be of particular interest to those interested in implementing the Keepaway Framework.

## 1.5 Definitions & Operational Terms

The following definitions have been provided to ensure uniformity and understanding of these terms throughout this study.

Action-centric: This term is derived from the general definition of "centricity" as "centring on or focusing on" (Collins English Dictionary, Sinclair, 2000, p132) which, in this case, implies centring or focusing on the action i.e. the action is central to consideration.

Action-Selection Policy: The method, or methods, used by an agent to determine the next course of action it wishes to perform within its environment.

Agent: The term "agent" is used as a short form of the term "autonomous agent", unless otherwise stated, which is defined as "a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." (Franklin et al, 1996, p3)

Collaborative Agent: Collaborative agents are an element of a society than can perceive, although often limited, aspects of its environment and affect that environment either directly or through cooperation with other agents (Luger, 2002).

Confidence Levels: This term is used to describe a degree of how confident, or certain, an agent is in the abilities of its peers and its opponents. This concept is discussed in detail in section 3.2.2.

Domain: "A sphere of activity, concern, or function; a field" (Collins English Dictionary, 2000, p247). Used in this context to describe the fields of activities which make up the field of multi-agent collaborative robotics.

Entropy: This term refers to the amount of diversity in a society or simply put: the amount by which the group is not similar. This definition is more inline with the definition of entropy found in the study of biodiversity and in Balch's work on entropy (Balch, 2000) rather than more technical definitions used in thermodynamics and information theory.

Heterogeneous: Although open to interpretation, heterogeneity is commonly described as being "composed of parts having dissimilar characteristics or properties" (Collins English Dictionary, Sinclair, 2000, p393).

Homogeneous: This definition of homogeneity is also open to interpretation, but simply described as: "uniform in structure or composition throughout" (Collins English Dictionary, Sinclair, 2000, p400).

Keepaway Framework: This term refers to the subtask of RoboCup Simulated Soccer, proposed as a machine learning test bed by Stone, Sutton & Kuhlmann (Stone et al, 2005a) and written by Stone, Kuhlmann, Taylor & Liu (Stone et al, 2005b). The Keepaway Framework is covered in detail in sections 2.2.4 and 3.2.1.2 of this study.

## 1.6 Delimitations

The following factors have been identified which delimit the scope of this study:

1. Due to the large number of multi-agent collaborative domains, in-depth investigation was limited to robot rescue and robot soccer with other domains receiving only cursory mention.

2. Due to the number of action-selection methodologies in use, investigation was limited to a selection of those observed to be the most popular in recent competitions and those with historical research significance.

3. The Keepaway Framework policies were chosen to compare the performance of the proposed action-selection approach against because of their facility for direct evaluation which was considered a more desirable quality over the implementation of any of the alternative policies documented which, while more realistic, lack any documented performance standards.

# 2. Collaborative Multi-Agent Domains

## 2.1 Introduction

The definition of what constitutes an agent seems to be entirely dependant on the application or characteristics that a particular researcher or group of researchers is studying.

Virdhagriswaran, the creator of the MuBot Agent defines an agent as a representation of "two orthogonal concepts. The first is the agent's ability for autonomous execution. The second is the agent's ability to perform domain oriented reasoning" (Virdhagriswaran, as cited in Franklin & Graesser, 1996). By this definition, autonomous execution is clearly a central component of agency.

Russell and Norvig, authors of the popular text, *Artificial Intelligence: A Modern Approach* state "an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors." (Russell & Norvig, 1995, p33) This text focuses heavily on software agents being used to explore artificial intelligence (AI) techniques. This definition clearly depends on the further classification of the terms "environment", "sensing" and "acting". If the environment was defined as a mechanism providing input and output, and take receiving input to be sensing and producing output to be acting, by this definition, every program is an agent. Thus, to arrive at a clear division between agents and programs, better restrictions must be put in place when classifying concepts of environment, sensing and acting.

Maes, on the other hand states: "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realise a set of goals or tasks for which they are designed." (Maes, 1995, p108) With this definition she adds a crucial differentiator: agents must act autonomously so as to "realise a set of goals." She also restricts the environments to being "complex and dynamic", although it is not clear if this definition omits more complex programs, such as some accountancy packages, from agent classification without further restrictions.

Hayes-Roth goes one step further with: "Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions" (Hayes-Roth, 1995, p329). From this definition, it is clear that there are more to agents than the ability to autonomously process inputs and outputs, they must also be capable of reasoning in order to select appropriate actions as well.

Wooldridge and Jennings set clear limits in their definition for what constitutes autonomy, sensing and acting. They also introduce a communication requirement and further reduce the scope of suitable agent environments. Hence, they define an agent as: "... a hardware or (more usually) software-based computer system that enjoys the following properties:

- Autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- Social ability: agents interact with other agents (and possibly humans) via some kind of agent-communication language;
- Reactivity: agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- Pro-activeness: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative." (Wooldridge & Jennings, 1995, p2)

As Franklin and Graesser found in their paper: *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents*, these definitions can be combined together to form the following general definition:

"An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." (Franklin & Graesser, 1996, p3)

Although boundaries for this definition could be determined by a study of the extreme cases, "the notion of an agent is meant to be a tool for analysing systems, not an absolute characterisation that divides the world into agents and non-agents" (Russell & Norvig, 1995, p. 33). In essence, all of the previous definitions of agents share two basic commonalities:

1. An agent has the ability to perform actions, and;
2. An agent is responsible for acting for another with permission.

(Franklin & Graesser, 1996)

This general definition reflects the general nature of agents, as agents can be defined in biological terms, as a type of virus, in legal terms, an agent is a person or entity that has a legal duty to act in another person's best interest and in linguistics, an agent is a noun carrying out an action (Wikipedia, "Agents").

It should also be noted that the term 'robot' is generally synonymous with the term 'agent' as a robot is a common form of agent seen in both hardware and software.

Going back to Wooldridge and Jennings' definition, in order to qualify as an agent, the entity must possess the ability to act autonomously. This is considered an important distinction because "a simple act of obedience to a command does not qualify an entity as an agent" (Wooldridge & Jennings, 1995, p2).

Agents, according to this definition, must therefore be either autonomous or semi-autonomous, for example, a household vacuum-cleaning robot can be considered autonomous in that it is dependent only on a human operator to start it up.

However, this requirement is a matter of some controversy among researchers as it can be argued that this level of autonomy is not in fact "true autonomy" as, in practice; all agents are under active human supervision. Human nature also dictates that, the more important the activities of the agent are to humans, the more supervision they receive. Given this definition it can be construed that true autonomy is seldom desired, and instead effort should be put into development of interdependent systems.

In many domains of agent research, such as exploring an unknown planet, search and rescue, or cleaning up toxic waste, it has been proposed that rather than sending one very complex agent to perform the task, it would be more effective to send a number of smaller simpler agents. Such a collection of autonomous agents is often described as a swarm, a colony or as a collective, as they often exhibit cooperative behaviour (Balch & Parker, 2002).

By definition, collaborative agents are an element of a society than can perceive, although often limited, aspects of its environment and affect that environment either directly or through cooperation with other agents (Luger, 2002).

Collaborative agents are most often used to form Multi-Agent Systems where several agents, interact through message passing or changes in their common virtual environment. These systems can also include human agents. By definition, human organisations and society in general can be considered an example of a multi-agent system.

Multi-agent systems are highly structured. Each agent has a set of skills and responsibilities, which it uses through coordination with its peers, to solve the given problem until a final solution is achieved. Thorough this process, Multi-agent systems can often manifest self-organisation and complex emergent behaviours even when the agent strategies used are simple (Luger, 2002).

This phenomenon of "emergent" intelligence comes to light as the overall cooperative result of the Multi-agent system, can be viewed as greater than the sum of its individual contributors. Intelligence is seen as a phenomenon resident in and emerging from a society and not just a property of an individual agent (Luger, 2002).

Hence agents exhibiting or capable of exhibiting such emergent behaviours are referred to as "intelligent agents" and are said to possess certain abilities, including:

- Reactivity: The ability to perceive their surrounding environment and respond quickly to changes.
- Proactivity: The ability to formulate and carry out goal-directed behaviour.
- Sociability: The ability to interact with other agents.

(Luger, 2002)

Intelligent agents are used to automate certain tasks by making choices in given situations based on rules. These rules differ depending on the domain the agent is being used in. Currently there are a number of intelligent agents commonly seen in the following domains:

- Buyer Agents: Also known as Shopping Bots. These agents assist internet shoppers by finding the products and services the user is searching for. The results of these agents are seen on sites such as Amazon and eBay which provide users with a similar or related products display. This listing is compiled by the agents from data based on viewing/shopping data gathered from the site and from the users own viewing and shopping habits.
- Data Mining Agents: This agent is used to find trends and patterns in large quantities of data and information from various sources and report its findings back to its user so they can make decisions accordingly.
- Entertainment Agents: These agents can take many forms from virtual pets such as Neopets which require a certain amount of attention and interaction from the user, to virtual opponents or 'bots', found in game series such as Half-Life, Counterstrike, Age of Empires and Neverwinter Nights.
- Information Agents: Also called 'information retrieval agents'. These agents retrieve and present information based on a person's local context. Three such agents have currently been implemented by MIT: the Rememberance Agent, Margin Notes and Jimminy. Many studies have shown that information retrieval agents allow users to become more efficient retrieving and using information than through traditional searching alone.
- Management Agents: Service management agents, manufacturing management agents, and business process management agents are designed to support business decision

making. Both service and manufacturing management agents, in addition, have a scheduling capacity and can potentially control the tasks assigned to other human and robotic agents within organisations.

- Predictive Agents: Also called Monitoring or Surveillance Agents. These agents observe and report on the operation of equipment or systems, most commonly computer systems. These agents are used to track inventory levels, competitor's prices, and most recently, stock market manipulation by insider trading.
- User Agents: Also called Personal Agents. These agents assist users by carrying out organisational tasks such as sorting emails according to the user's preference, assembling news reports or other information of interest or filling out webpage forms with stored information. These agents can also act as spam filters and search indexers.

(BotSpot.com)

It has been argued that many of the intelligent agents discussed above are actually better classified as knowledge-based or expert systems, or even just as 'programs'. Expert systems are defined as: "a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice" (Jackson, 1999, p2). However, from an agent's perspective, there are in fact a number of fundamental differences between them and knowledge-based systems, including:

- Expert systems are not coupled to their environment and hence exhibit no knowledge of their physical reality.
- Agents are capable of both reactive and proactive behaviours.
- Expert systems are not capable of acting collaboratively in societies.

(Franklin & Graesser, 1996)

On the other hand, looking at the characteristics that are supposed to differentiate an expert system from "other kinds of artificial intelligence programs" (Jackson, 1999, p3), it can easily be seen how they are often confused. Expert systems must:

- Deal with subject matter of realistic complexity that normally require a considerable amount of human expertise;
- Exhibit high performance in terms of speed and reliability in order to be useful; and
- Be capable of explaining and justifying solutions or recommendations in order to convince the user that its reasoning is in fact correct.

(Jackson, 1999, p3)

The author justifies these characteristics by stating:

- Many AI programs deal only in 'toy worlds' and expert systems deal with problems of genuine scientific and commercial interest;
- "Most AI programs are buggy and run slowly whereas expert systems must be as timely in their responses as human experts; and
- Expert systems will be run by a wider range of users, rather than researchers in a lab, and therefore should be designed so that its workings are transparent.

(Jackson, 1999, p4)

These justifications, however, appear terribly narrow minded, given the date of publication and the wide range of applications that agents were used for since becoming the 'in thing' in the latter half of the nineties.

In fact, agents have been put to many uses not only in software as discussed above as shown in taxonomy below:

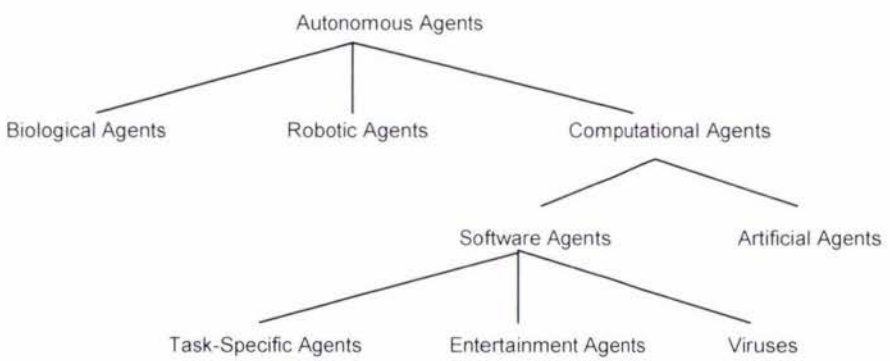



Figure 2.1: A Taxonomy of Autonomous Agents (Franklin & Graesser, 1996)

In order to foster greater interest in multi-agent research projects, a number of "grand challenge" problems have been proposed including, but not limited to: robot soccer and robot rescue initiatives, robot swarm or distributed intelligence initiatives, and autonomous vehicle development.

The term "grand challenge" was originally coined by The Defence Advanced Research Projects Agency (DARPA) in an effort to create interest in the development of technologies capable of assisting the armed forces, but has been adopted by many agencies as a term referring to an area of research which is of interest to both the general public and academia.

As the most well known and well regarded collaborative multi-agent domains, the next sections will look in detail at robot soccer and robot rescue as potential areas in which to investigate the role of heterogeneity in robot teams.

## 2.2 Robot Soccer

A soccer game is a specific but very attractive real-time multi-agent environment and promotes research into distributed artificial intelligence, network based multi-agent interactions, computer graphics, computer vision and physically real animation.

From a multi-agent perspective, robot soccer is an interesting research domain as it investigates collaborative and emergent behaviour within a competitive environment. Each team has a team-wide common goal, to win a game; the goals of the two teams are incompatible and the opponent team can be seen as a dynamic and obstructive competitor; which hinders the agents achievement of the common goal.

Agents will therefore need to react quickly, flexibly and cooperatively in order to win the game. The team will also need to break this common goal into a number of both global and local strategies and tactics. These would include real-time planning, opponent observation and modelling, game tactics such as use of formations, roles within teams, and coaching, amongst others.

The popularity of this research domain is apparent as the organisations involved have provided an excellent framework which allows researchers to get up to speed quickly and at a reasonable cost. There are also provisions to promote research in school and university (or college) aged students.

There is also a certain amount of mystique surrounding soccer playing robots as it is considered a grand challenge problem and as such, will attract a certain amount of interest, regardless. Also it is an appropriate research avenue for burgeoning humanoid robotics development as it provides an opportunity for researchers to compare their approaches relating to gait, balance, and vision in action.

Although this research domain is accessible it also includes a number of unique challenges:

- The game environment is highly dynamic and constantly changing as the players and the ball manoeuvre around the field.
- The perception of each player is locally limited.
- The role and hence the responsibilities of each player is different.

- Communication with other players is limited, therefore each agent is required to behave very flexibly and autonomously in real-time under the resource bounded situation.

(Kitano, 1997)

### 2.2.1 Robot Soccer Initiatives

#### 2.2.1.1 RoboCup Soccer

"RoboCup is an international initiative devoted to advancing the state of the art in AI and robotics" (Kitano, 1997, p1).

The first mention of soccer playing robots was presented by Professor Alan Mackworth from the University of British Columbia, in a paper entitled "On Seeing Robots", at VI-92. Mackworth used the metaphor of a soccer playing robot to draw attention to the limitations of Haugeland's 'Good Old Fashioned Artificial Intelligence Robotics' (GOFAIR) paradigm proposed in his book *Artificial Intelligence: The Very Idea* (1985) (Mackworth, 1993) and (Haugeland, 1985).

The concept was again raised at the Workshop on Grand Challenges in Artificial Intelligence, organised by an independent group of Japanese researchers, in October 1992. This workshop led to a serious investigation of using the game of soccer to promote science and technology and included the development of prototypes of soccer robots, simulator systems and draft rules and regulations.

From the positive result of the investigations into robot soccer, a group of Japanese researchers launched a robotic competition, tentatively called the 'Robot J-League' after the Japanese Professional Soccer League. However, once announced, they received an overwhelming response from researchers worldwide to extend the competition and accordingly, the competition became the 'Robot World Cup Initiative' or RoboCup for short. The most ambitious and long range goal of the newly formed RoboCup can be stated as: "to build a team of robot soccer players, which can beat a human World Cup champion team" (Kitano, 1997, p1).

Also during 1993, Itsuki Noda, at the ElectroTechnical Laboratory (ETL), a government research centre in Japan, was conducting multi-agent research using soccer and started development of a dedicated simulator which later became the official soccer server of RoboCup (Kitano, 1997). Version 0.0 of the Soccer Server, written in Lisp, was released later that year followed by Version 1.0, a C++ version.

The first official RoboCup games and associated conference were held in 1997 at the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97) in Nagoya, Japan, as part of IJCAI-97's special program, with great success. Over 40 teams (real and virtual) participated and over 5000 spectators attended. (Kitano, 1997).

419 teams from 35 countries participated in the RoboCup 2005 competition held in Osaka, Japan in July. The competition comprised of 10 leagues in three divisions: RoboCup Soccer; RoboCup Rescue and RoboCup Junior. The leagues for RoboCup Soccer are described below:

1. Humaniod League:

Introduced in 2002, the Humanoid League competition consists of a penalty kick, and "1 on 1", and "2 on 2" competitions. The robots that take part range in size from 10cm to 2m and are separated into kid size and mid size divisions. Initially human intervention was allowed as some robots were tele-operated, but this practice has now been discontinued.

2. Small-size Robot League:

To qualify as Small-size, robots must be no higher than 150mm and must fit within a 180mm diameter circle. They also have a unique choice of using either local or global vision systems, although global vision seems to be preferred due to the small size of the robots. Games consist of 5 robots aside and are played on a 4.9m x 3.4m field with an orange golf ball.

3. Middle-size Robot League:

To qualify as Middle-size, robots must be 80cm high, and between 30 and 50cm wide on any side. Depending on the size of the robots used, teams can consist of 4 or 5 larger robots or 6 smaller robots as long as the total size of the team does not exceed 10,000cm$^2$. Games are conducted on a 12m x 8m field using an orange size 4 regulation soccer ball.

4. 4-legged Robot League:

In the 4-legged league, teams consist of 4 Sony AIBO ERS-7 or ERS-7M2 robots. Equipped with onboard sensor and vision systems, teams play on a 5.4m x 3.6m field with an orange tennis-sized ball. The robots are expected to act autonomously with no outside communication or control permitted, except for the game computer. Along with soccer games, teams are

expected to compete in a number of technical challenges designed to put the teams AI's to the test.

5. Simulation League:

The Simulation League has recently expanded to include three leagues of its own: the 2D Soccer Simulation League, 3D Soccer Simulation League and Soccer Simulation Coach League. The 2D and 3D leagues are fairly similar in that games consist of 2 teams of 11 virtual agents competing against each other on a simulated soccer field that provides a realistic simulation of soccer robot sensors and actions, with the only difference being that one takes place in 2D space and one in 3D space. In both cases, games are 10 minutes long and consist of two 5 minute halves. The Coach league however involves pitting coaches (or directors) against each other by comparing their performance managing a variety of teams. The focus of this competition is on team and opponent modelling and online adaptation. Coaches have the ability to work offline by analysing logs of previous games or online by adapting team behaviour while the game is proceeding.

(RoboCup, 2005)

Although in the last few years, RoboCup has expanded its goals to include rescue initiatives and opportunities to foster interest in robotics with the researchers of tomorrow, RoboCup is still primarily a robot soccer initiative and still receives most of its public interest from that quarter. Spectators flock to watch the final games and the open qualifying competitions held annually around the world.

#### 2.2.1.2 FIRA


Figure 2.2: FIRA logo and examples of participant classes (FIRA, 2005).

FIRA Robot Soccer began in 1995, with the first international championship taking place in Daejeon Korea, 1996. The Federation of International Robot-soccer Association was founded in 1997 by Professor Jong-Hwan Kim, with the basic goal of "taking the spirit of science and technology of robotics to the laymen and younger generation, through the game of robot soccer" (Kim, 2005).

The FIRA Robot World Cup is held every year with participants from around the world. The FIRA Robot World Congress is held alongside the FIRA Cup competition, where all participating teams

will submit and present papers on their robots and schemes for sharing the expertise and technology for building the soccer robots.

FIRAs focus is primarily on physical multi-agent research and hence focuses closely on the underlying mechanics of a robot team and basic skill acquisition with a lesser focus put on AI development, hence, five of six classes of soccer player are dedicated to various types of hardware as pictured in figure 2.2 above. These classes are:

1. HeuroSot

A humanoid type robot soccer player of a maximum height of 150cm and maximum weight of 30kg. This category encourages research into walking and balancing, complex motion planning and human robot interaction. The events in this competition include: robot dash, obstacle run, penalty kicks, lift and carry, and "Where's Waldo" (also called "Where's Wally").

2. KheperaSot

This competition is played between two Khepera robots on a 1 on 1 basis. Games are played on a 1.3m x 0.9m pitch with a yellow tennis ball. As the Khepera robots are equipped with an on-board vision system and are expected to be fully autonomous with human team members only allowed to place their robot on the field and start their robot at the beginning of each round at the position indicated by the referee.

3. MiroSot

To qualify in the "micro" tournament, the robots must be smaller than 7.5cm x 7.5cm x 7.5cm. Games are played on a 3 on 3 basis with an orange golf ball on a 1.5m x 1.3m pitch. Only one host computer may be used per team and is often dedicated to vision processing. As one of the fastest tournaments, with robots capable of speeds of 2m/s, much consideration is put into ensuring that the body of the robot is able to withstand potential "high-speed" impacts, the equivalent of dropping the robot from waist height onto a hard floor.

4. NaroSot

Measuring no more than 4cm x 4cm x 5.5cm, the NaroSot robots compete on a 5 aside basis on a 1.3m x 0.9m pitch with an orange ping pong ball. Because of their small size, NaroSot competitors tend to make use of the global vision system option during the tournament. Similar

to the MiroSot competition, the NaroSot teams are only allowed to use one computer which tends to be dedicated to vision processing and localisation.

5. RoboSot

With an option for operating either fully or semi-autonomously, the RoboSot competitors must be no larger than 20cm in width or length, but have no restriction on height. Teams of between 1-3 robots play on a 2.6m x 2.2m pitch with a yellow and green tennis ball. No form of central computer or global vision system are allowed, early development primarily focused on on-board sensor, actuator, communication and control development before the exploration of intelligent or autonomous systems.

6. SimuroSot

Two teams compete against each other using the SimuroSot server. Client teams can either represent mid or large size robots and they play against each other in either 5 aside games and 11 aside games. Removed from hardware constraints, this tournament primarily focuses on team strategy and control.

(FIRA, 2005)

With a primary focus on hardware development and basic soccer skills acquisition, FIRA shares many similarities with RoboCup. But with a narrower scope and a smaller number of dedicated leagues, FIRA's focus remains firmly on studying multi-agent teams in a highly dynamic and uncertain domain.

### 2.2.2 Robot Soccer Simulators

Simulation has always been an important feature of any endeavour where expensive equipment and human safety are involved. Simulators allow researchers to safely test out their theories and view the results in a safe environment.

However, conducting research in a simulated environment raises issues about its suitability when applied to real hardware in the real world. Most simulators try to address this issue by staying as close to the true physicality of the scenario as possible. This can be seen in the limitations in auditory and visual data used in the RoboCup Soccer Server which help recreate the real conditions experienced on a field and in its use of stamina so that players cannot run around the field for unreasonably long periods of time at full speed.

Currently there are a number of simulators used to simulate robot soccer, as well as the official simulators of the Roboup and FIRA leagues, other popular choices include TeamBots, which is considered superior for AI development and Webots which can simulate games between Sony AIBO.

#### 2.2.2.1 RoboCup Soccer Server

The Soccer Server is the official simulator used in the RoboCup Simulator League. The RoboCup Soccer Server started as a "network-based, graphical simulation environment for multiple autonomous mobile robots in a 2D space" (Kitano, 1997, p9) and since 2004, has been updated to support both 3D graphical and physical representation. The simulators primary contribution to the goal of RoboCup, is to support comparison among multi-agent systems by providing a framework to test how well agent cooperation techniques work in a dynamic environment.


Figure 2.3: RoboCup Soccer Server.

All games are visualised by the Soccer Monitor, as shown in figure 2.3 above, a separate application which connects to the Server and displays the current state of the field on a computer screen. Client programs, each representing a soccer player, also connect to the server via UDP/IP. Once connected, that client controls the player it represents on the soccer field. All movements and collisions of the players and the ball are simulated in a simplified form to enable real-time execution (Chen et al, 2003).

One of the advantages of the Simulator League over its physical counterparts is its level of abstraction, which saves researchers from having to worry about so-called lower-level hardware-based problems such as object recognition and tracking, mobility and balance, and sensing and communications. This abstraction enables researchers to focus on higher-level concepts such as co-operation and learning. In fact it is widely acknowledged within the RoboCup community that the simulation league boasts the best teamwork and cooperation of any of the leagues which can be seen in the number of pass combinations in front of the goals.

This increased level of abstraction has also created its own problems as "the intentions of the players cannot mechanically be deduced" (Dudek et al, 2002, p21) so other methods have to be put in place to ensure a level playing field. This includes a two-fold refereeing scheme which sees a virtual referee, responsible for enforcing basic play rules such as offside and scoring, working alongside a human referee, responsible for enforcing harder-to-detect situations such as deadlocks caused by crowding. All participating teams are also obliged to play according to a gentlemen's agreement and not exploit loopholes created within the simulation system, although these loopholes have become scarcer as the system has evolved.

With the introduction of a coaching system to the Server, teams now have the additional flexibility of using an online coach to control team strategy during a game. This comes with another recent addition to the Soccer Server, the ability to support heterogeneous players, which allows teams to be comprised of players from 7 distinct types and allows the coach to substitute a maximum of 3 players for one of a different type during the game. These player types are determined by their profile which specifies the maximum values that the "player variables" can reach including speed, stamina, and power (Chen et al, 2003).

While one of the most advanced simulators in terms of physics and accurate player modelling, the Soccer Server is still a low-level simulator requiring users to write their own low-level playing skills such as pass, kick and dribble before they can form a basic usable team. This often acts as a deterrent for researchers who simply want to use the simulator as an AI or multi-agent test bed and not actually create a competitive team.

#### 2.2.2.2 SimuroSot 3D Soccer Simulator

The SimuroSot 3D Soccer Simulator is the official simulator of the FIRA Robot Soccer League. The aims for the simulator include providing a platform for development of basic action algorithms and game strategies for robot soccer and to test the feasibility and advancement of the game strategy of each team.


Figure 2.4: Screenshot of the SimuroSot Simulator in action.

To achieve these goals, the simulator uses a combination of computer graphics to simulate the field, the robots and the ball and kinematics and dynamics to simulate the movement of the ball and players.

Similar to the RoboCup Soccer Server, the SimuroSot system consists of a server which hosts the game environment which the players or clients then connect to. Each team's client program, written in C++, connects to the server using the UDP/IP protocol and then the running of the game becomes the responsibility of the human referee.

Unlike the RoboCup Soccer Server, the SimuroSot simulator is capable of simulating two player sizes, representing medium and large sized robots played 5 or 11 aside. The game is played over two 5 minute halves with the option of a 3 minute sudden death period if the game ends in a tie. Another feature of this simulator is that it allows each team two timeouts during the game for teams to 'substitute' their strategies.

Although the SimuroSot simulator has been further advanced graphically, than the RoboCup Simulator, it is less advanced in terms of its real-worldness. As client teams are presented in a single file, it would be impossible to police communication between players and make the necessity for aural sensing redundant which would prove a hindrance if the simulated controllers were to be transferred to real robots without the ability to use some form of shared memory.

#### 2.2.2.3 TeamBots

TeamBots is a collection of open-source Java packages designed to assist research for mobile multi-agent robotics. Although some of the lower-level functions are handled in C through the use of the JINI interface, all higher level functions are written in Java (Blach, 2000b).

TeamBots allows developers to quickly prototype multi-robot control systems which can be tested in a simulated environment before being run on physical robots without needing to be ported or changed in any way. It is capable of supporting multiple heterogeneous robot hardware running heterogeneous control systems. Experimental environments containing walls, roads, opponents and circular obstacles all of which can be altered from a human-readable description file.

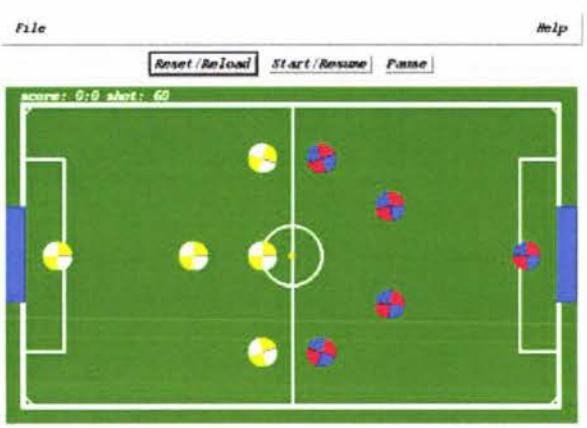



Figure 2.5: Screenshot of SoccerBots showing set kick-off positioning (Balch, 2000b).

TeamBots has a number of pre-configured domains, one of which is SoccerBots, configured for soccer playing mobile robots. SoccerBots simulates the dynamics and dimensions of a regulation RoboCup small size league game. Two teams of five robots compete on a simulated ping-pong table by pushing and kicking an orange golf ball into the opponent's goal (Blach, 2000b).

At TeamBots present development stage there is no facility for communication between the robots and no support for a specialised "goalie" which are key features of other simulators. Also the kick-off positioning is fixed in the formation shown in the above figure which would limit the assistance of any kind of coaching agent as these kick-off formations have been shown to be key elements in game strategy.

TeamBots provides an array of sensors that provide a series of vectors towards the ball, the teams goal and the opponents goal as well as vector arrays pointing just to the robots team and

opponents. Other egocentric inputs include the players' number on the team, position, heading and the elapsed game time in milliseconds.

Outputs are achieved through simulated actuators which enable the robot to kick the ball at 0.5 metres per second, push the ball by moving into it, turn at 360 degrees per second and move at 0.3 metres per second (Balch, 2000b).

Although not as evolved in terms of game realism as either the RoboCup Simulation Server or the FIRA SimuroSot simulator, TeamBots does allow users to get up to speed quickly by providing basic game behaviour primitives such as run, kick and pass which could account for its popularity as a multi-agent research tool.

#### 2.2.2.4 Webots

Developed by Cyberbotics Limited, Webots is a mobile robotics simulation package that provides a rapid prototyping environment for simulating mobile robots. Included real robot models consist of the Sony AIBO, Fujitsu HOAP-2, Khepera, Alice, Koala, Moorebot and Pioneer2.

Another advantage of Webots is that it allows developers to write their control system in a familiar language such as Java, Lisp or Pascal and connect to the simulation environment by TCP/IP (Cyberbotics, 2004).

Figure 2.6: Simulated soccer game between two Sony AIBO robots (Cyberbotics, 2004)

Webots would be a good choice for researchers who participate in both the FIRA and RoboCup competitions and want to test out their control theories in a virtual environment as the incorporated simulation models include both the Sony AIBO, used in the RoboCup Four-legged League, shown above in figure 2.6, and FIRAs Mirosot meaning they wouldn't have to run two simulation environments.

As alternative simulators tend to concentrate more on higher level behaviours, they are not easily transferable onto real hardware. However a controller developed using Webots could, according to their documentation, be easily transferred and run on a real AIBO.

This is not to say however, that Webots is an unsuitable simulation environment for testing theories of higher level multi-agent behaviours. In fact, the diversity of the provided models and the attention given to both the graphical and physical representations of the simulated environment will ensure that it remains a worthy choice for those wishing to get up to quickly simulate a particular robot without having to first reinvent the wheel.

### 2.2.3 Real Robot Soccer

Although the goal of robot soccer is to create a team of humanoid robot players capable of playing (and hopefully beating) the current world champion human soccer team of the time, researchers are using cheaper and more manageable ways to test out their theories on smaller platforms before amalgamating the best of these theories together and scaling up to the final solution.

Each size and type of robot has a number of similar and/or unique capabilities that make it the perfect test bed for the development and investigation of certain aspect(s) of the soccer domain, which need to be fully explored before a team of humanoid players can be fielded.

Small sized robots, illustrated in figure 2.7, allow researchers to explore more strategic elements of the game including multi-agent collaboration, real-time planning, reactive behaviour, strategic decision making, opponent modelling and ball handling. As they commonly use a global vision system, more processing power can go to analysing the tactics of the other team and preparing team strategies.

Figure 2.7: Small-sized team at RoboCup 2005.

Medium sized robots, with onboard vision systems allow researchers to explore vision related problems such as self-localisation, feature recognition, object discrimination and tracking, and perception along with the strategic elements tested with the smaller sized robots. Also with the opportunity of creating a team of heterogeneous robots, researchers can explore the effect of trade-offs with the size of players, the number of players on the team and their configurations in terms of speed and kicking power.


Figure 2.8: Medium-sized team at RoboCup 2005.

Four-legged robots, as shown in figure 2.9 below, allow researchers to concentrate on developing their AI systems as the hardware for the teams is standardised with onboard vision systems and touch sensors. As well as providing a good base for higher level cognitive development, the four-legged players must still have a good implementation of vision, movement and ball handling skills. Player recognition also plays an important part as the robots look so similar. Areas which have been recently researched using four-legged robots include task decomposition and role allocation.

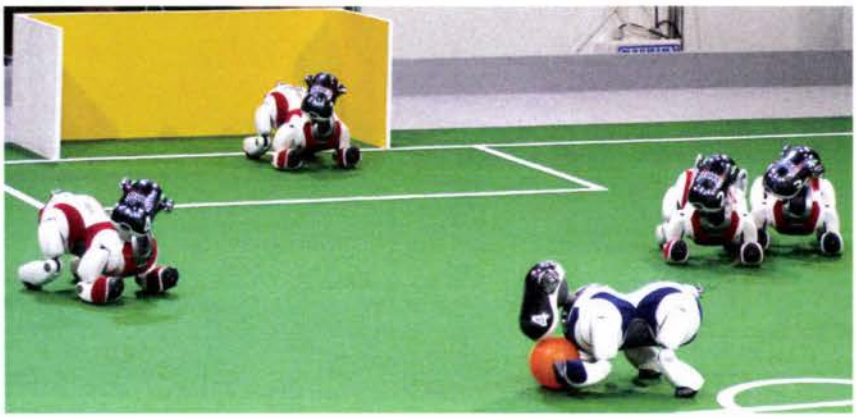


Figure 2.9: Four-legged team competition at RoboCup 2005.

Humanoid robots are expected to be able to walk using 2 legs (no training wheels allowed), and exhibit basic skills such as body control, balance, shooting a ball, defending a goal and distinguishing relevant objects by colour. On top of perfecting movement, in order to play soccer, humanoid robots must be capable of ball handling, communication and cognitive modelling. These research areas in turn require research into higher cognitive functions such as object discrimination and tracking, opponent observation, self-localisation, map building, and spatial perception.

Figure 2.10: Humanoid robot from team Osaka getting ready to compete at RoboCup 2005.

Finally, all of the robot types must have the ability to withstand contact from other players during the course of a game and aim to minimise any safety concerns arising from their close contact with human players.

### 2.2.4 Soccer Keepaway

Soccer keepaway is a sub-problem of the RoboCup simulated soccer domain in which one team, the keepers, tries to maintain possession of the ball within a bounded region, while the opposing team, the takers, attempts to gain possession. Whenever the takers gain possession or the ball leaves the region, the episode ends and the players are reset for another episode (with the keepers being given possession of the ball again). An example episode is shown in figure 2.11 below. Parameters include the size of the region, the number of keepers and the number of takers. The game is controlled by an omniscient coach agent responsible for managing play, ending episodes and resetting the field. (Stone et al, 2005)

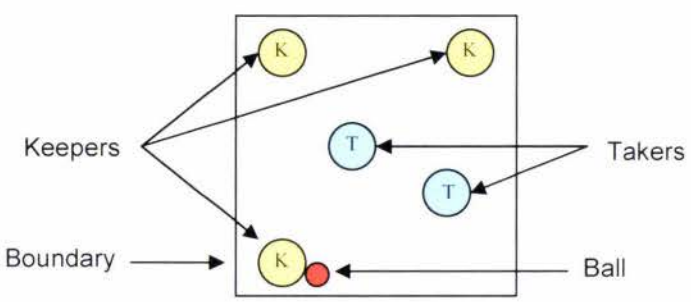



Figure 2.11: Example of an episode with 3 keepers and 2 takers playing in a 20m x 20m region.

This sub-domain has now been put forward as a test-bed to facilitate machine learning development that provides a repository that:

1. Provides standard, open source implementations for all aspects of the problem except the learning algorithm itself;
2. Provides a step-by-step tutorial for non-domain-experts to get up to speed easily; and
3. Provides the graphical tools necessary to evaluate progress.

(Stone et al, 2005)

Another advantage of this sub-domain over its parent domain is that it is possible to compare concrete numerical results between tests by evaluating the average possession time the keepers maintain hold-of of the ball.

The following benchmark results were given by Stone, Kuhlmann, Taylor and Liu for three static keepaway policies. The 'Always Hold' policy dictates that once a keeper has possession of the ball it maintains sole possession of it and does not pass. The 'Random' policy sees the keepers randomly select one of the available actions. Finally the 'Hand-Coded' policy is a simple strategy that compels the keeper to hold the ball when no takers are nearby otherwise they determine their

most open team mate and pass to them. Results are measured in seconds and displayed with their standard deviations.

This Framework has already been used in successful studies using temporal difference reinforcement learning with function approximation by Stone, Sutton and Kuhlmann; evolutionary learning by Di Pietro, While and Barone; relational reinforcement learning by Walker, Shavlik and Maclin; and behaviour transfer by Taylor and Stone.

## 2.3 Robot Search and Rescue

Robot search and rescue, similar to robot soccer, promotes collaboration between the physical and software aspects of robotics to form teams capable of recovering victims in the event of a natural or man-made disaster. However unlike robot soccer, these teams include human operators, either to control tele-operated components or to retrieve victims located by the robots and intelligent software components capable of accessing building blueprints and hazardous materials information over the internet.

The robots comprising these search teams tend to be heterogeneous in nature with many specialist types designed to carry out specific tasks. Some robots are equipped with microphones to detect voices, breathing or other sounds of human presence, others are equipped with thermal cameras to detect body heat, or high definition cameras to detect traces of colour amongst the dust blanketed debris. Other than their various sensors, robots differ in their physical abilities, such as the ability to climb stairs or burrow through rubble.

With a window of opportunity of only 48 hours until a rescue mission becomes a recovery mission, there is an obvious need for small and cheap search and rescue robots that can be deployed into a disaster area by the hundreds to locate victims.

As Robin Murphy and her research team found at ground zero, following the attacks of 9/11 most of the robots they trialled were "not yet sophisticated enough to roam the rubble." The researchers noted that "the 'hot zone' of the rubble is in vertical piles, which the robots are not capable of climbing" (Trivedi, 2001).

However, the experiences of New York, have given Murphy and her team insights that will help them design faster, smarter and more independent search and rescue robots, which will be critical to optimise the number of rescues in that 48 hour period (Trivedi, 2001).

Currently much work has been done in the development of both physical robots capable of entering buildings under a number of conditions and simulators and software applications capable of supporting these robot rescuers.

### 2.3.1 Robot Rescue Initiatives

#### 2.3.1.1 RoboCup Rescue

The basis of RoboCup Rescue is a disaster rescue scenario in which teams of rescue agents attempt to minimise civilian casualties and building damage after an earthquake. Agents in rescue teams carry out rescue operations in a simulated disaster world, and must cooperate with each other to save trapped and buried victims, extinguish fires, and to repair and clear roads etc.

Teams do not compete against each other directly like games in RoboCup Soccer instead they asynchronously tackle the disaster environment and are graded on the number of victims found and the accuracy of the information gathered about the victims and their locations (Takahashi, 2003).

The agents involved in Rescue are inherently heterogeneous with different abilities requiring at least implicit cooperation if the system as a whole is to be effective. There are however many questions as to how this cooperation can best be achieved. It is important that methods chosen are robust so that if some part of the system is rendered inactive, the rest of the system continues to operate in a reasonable, if not optimal manner (Padgham et al, 2002).

The RoboCup Rescue competition comprises of two main parts: a simulation project; and a rescue robot project.

Competitors in the simulation project design a set of software agents which assume the roles of the various actors in the disaster mitigation process including rescue teams, fire fighters, police officers, ambulance officers and their department support staff. (Kuwata et al, 2001)

In the rescue robot competition, teams design and build a mobile robot capable of entering buildings that have collapsed or are near collapse, looking for survivors. The robots wouldn't pull anyone to safety, but would try to locate the living and show human rescuers where to find them.

Like robotic soccer, it's an application that requires a robot to be a team member. But a rescue robot's team would include human operators, as well as "intelligent software agents" that could automatically find information from the Internet and other databases about building blueprints, hazardous materials, or the occupants themselves.

#### 2.3.1.2 NIST

The National Institute of Standards and Technology (NIST) have developed and maintain a number of arenas used not only for direct competition, but also for testing. There are three types of arena: yellow, orange, and red, the colour denoting the level of difficulty.

Since 2000, NIST have hosted a yearly competition allowing researchers to compare physical robot solutions in a rescue scenario simulating a partial building collapse due to an earthquake.

The goal of the competition is to "minimise risk to search and rescue personnel, while increasing victim survival rates, by fielding teams of collaborative robots" (Messina, 2003). These collaborative robots are expected to assist human rescuers to quickly locate and extract victims by:

- Autonomously negotiating compromised and collapsed structures
- Find victims and ascertain their conditions
- Produce practical maps of their locations
- Deliver sustenance and communications
- Identify hazards
- Place sensors to monitor the situation (i.e. acoustic, thermal, hazmat, and seismic)
- Provide structural shoring

(Messina, 2003)

In this scenario, the robots (and their operators) are considered to be under the control of the Incident Commander, in charge of the disaster site, and report their findings directly back to him or her. Teams are then graded on the number of victims found and the accuracy of their interior maps and victim data sheets.

In order to achieve a good result, teams have to concentrate on many development elements including collaboration, autonomy, mapping and planning, knowledge representation, sensory perception, human-robot interaction and most importantly locomotion.

Many of these elements are synonymous with robot soccer however to be a successful rescue team, more emphasis is put on evolving collaboration amongst the heterogeneous rescue agents, rather than typically homogenous soccer agents. Emphasis is also put on developing systems to streamline the interaction between human operators and rescue robots. Locomotion is also focused less on "legged" solutions such as bi-pedal and quadra-pedal solutions and focused more on better suited configurations for the terrain involving wheels and treads.

NIST provides a simulator based on the Unreal Tournament engine, a popular 3D gaming platform, which allows researchers to accurately model their robots and simulate runs through photo-realistic representations of the three search arenas. However the simulator is only used as a research tool and the competition does not include teams competing in a virtual disaster area.

### 2.3.2 Robot Rescue Simulation

#### 2.3.2.1 RoboCup Rescue Simulator

The initial stages of RoboCup Rescue initiative were concerned with building a simulation environment chiefly for research and competition, while the eventual goal remains to support disaster management before and after a major catastrophe.

The RoboCup Rescue Simulation project creates a virtual large urban disaster scenario, usually the aftermath of a major earthquake, which is composed of various kinds of resultant conditions such as fires, building collapses, road blockages, dazed and confused civilians etc as shown in action in the figure below:


Figure 2.12: RoboCup Rescue simulator in action at RoboCup 2005.

The simulators for these states have been developed independently and are plugged into the Rescue Simulation kernel along with the various rescue agents developed by the competing teams. This ensures greater realism as the effects of each of theses conditions, such as fire spread and survivor evacuation, can continue to propagate through and effect the whole simulation system independently of the "human level" that the agents work at for the duration of the 72 hour period it covers (Takahashi, 2003).

Although the Soccer Server and the Rescue Simulator share many similarities, the Rescue Simulator must be able to handle a much greater number of heterogeneous agents as they cannot assume that their team mates share the same abilities as they do. Hence they must have some way of modelling their fellow agents. Sensory information is also much more limited in the rescue environment, so agents must be more active in acquiring information. The strategic component of the simulation also takes place over a much longer term. These differences are summarised in the following table:

| | Rescue | Soccer |
|---|---|---|
| Number of Agents | More than 100 | 22 |
| Team Composition | Heterogeneous | Homogenous |
| Information | Severely limited | Partial |
| Strategy | Long Term | Short or Middle Term |
| Goals | Many | Only One |

Table 2.1: Differences between the RoboCup Rescue Simulator and the RoboCup Soccer Server (Ohta et al, 2001)

### 2.3.3 Real Robot Rescue

In order to operate in a disaster environment, search and rescue robots must be weather proof and water proof in case they need to be decontaminated after coming into content with hazardous materials; they must be safe around humans; rugged so they are not crushed in the event they become trapped; and self-rightable so that they are not immobilised by being turned over (Murphy et al, 2001).

One potential role of these mobile robots is to provide an initial survey of the disaster site without risking the lives of rescuers or triggering a further collapse.

Murphy and her team have identified the following four roles rescue robots can assume:

1. Reconnaissance and site assessment
2. Rescuer safety
3. Victim detection
4. Structure characterising and mapping.

(Murphy et al, 2001)

Unlike robot soccer where the emphasis is on developing human-sized bi-pedal robots, rescue robot developers maintain a diverse number of differently designed, sized and configured robots as no one robot will be perfect for every rescue scenario.

For example many teams use robots with wheel or track based locomotion systems as these are well suited for traversing the rescue domain, and other teams use locomotion systems taken from biological ideas, such as the serpentine design of Carnegie Mellon's Snakebot, which can squeeze into hard to reach places. "There are different types of rubble, so there is no perfect robot for search and rescue. Some existing robots, like those in use in Afghanistan, function well in open space, or can climb stairs. But these snakebots could get into very confined spaces without further disturbing unstable rubble. It would be very effective and it's very exciting." (Handwerk, 2003)

Although the rescue competition has not been in operation as long as robot soccer, each competition gives developers a better understanding of the domain and the issues surrounding robot rescue, some of these issues include:

- Debris such as ropes, strings, newspapers, towels and futon mats can obstruct the motion of robots using crawler mechanisms.
- The wireless communication sometimes caused serious problems both with the transmission of camera images back to the operators, which often meant the operators where unable to move the robots, and on the occasion that the IP connection was cut due to the unstable wireless environment.
- Localisation proved to still be an issue as robots often lost their way in the rubble and often found the same victim many times.
- The skill of the operators also had a major impact on performance as the interface for the robot control has yet to be perfected.

(Tadokoro, 2003)

## 2.4 Summary

In order to be considered an agent, the candidate must possess a number of qualities, including the ability to act autonomously, perceive or sense its surrounding environment and act upon it, conduct goal-directed behaviour, reason, act sociably as part of a team, and act pro-actively. These characteristics are best summed up in Franklin and Grasser's definition of an agent: "An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future." (Franklin & Graesser, 1996, p3)

Collaborative agents have been used within a number of domains including a number of "grand challenge" problems including robot soccer and robot rescue initiatives which were discussed in detail.

Robot soccer is an interesting research domain as it investigates collaborative and emergent behaviour within a competitive environment. Each team has a team-wide common goal, to win a

game; the goals of the two teams are incompatible and the opponent team can be seen as a dynamic and obstructive competitor; which hinders the agent's achievement of the common goal.
So far physical player development has tended towards homogeneity, with teams sharing common software and hardware elements, often with minor changes made to a dedicated goalie. However with the number of simulation platforms available and their suitability as a test bed for AI development, much research is being completed around the strategic aspects of the game such as real-time planning and strategic decision making.

Both the RoboCup and FIRA soccer initiatives attract much attention from industry, academia and the general public with their diverse annual competitions and conferences. Although robot soccer is more entertaining than robot rescue, and let's face it, robots falling over in amusing ways in pursuit of a ball is a pretty good crowd puller, the reality of what the robot rescue developers are trying to achieve and the real-worldness of the problem, like reality television, make compelling viewing.

Robot rescue is a disaster rescue scenario in which teams of rescue agents attempt to minimise civilian casualties and building damage after an earthquake. Agents in rescue teams carry out rescue operations in a simulated disaster world, and must cooperate with each other to save trapped and buried victims, extinguish fires, and to repair and clear roads etc.

As there is no perfect agent, capable of handling all possible eventualities and environments within this scenario, rescue teams tend towards heterogeneity with teams using different software and hardware components and combinations and often including human team members as well.

Although robot rescue is a relative new-comer to the grand challenge domain, it still attracts a large amount of interest from industry, academia and the general public, not only through annual competition, but also through media documented research carried out after the attacks of 9/11.

Like robot soccer, robot rescue is still primarily focused on hardware, and the numerous problems associated with meeting autonomous agent criteria, such as developing perception, locomotion and collaborative skills. The primary AI development in robot rescue, however, is being put towards disaster management in the form of scheduling rescue efforts and resources intelligently.

Although, the full effects of heterogeneity in the rescue domain have not been fully investigated, I have concluded that RoboCup Soccer's sub-domain of keepaway would make a perfect basis for this study because of its smaller scope, quick setup time and facility for direct evaluation. These qualities are considered more desirable over the implementation of any of the alternative policies which, while more realistic, lack any documented performance standards.

# 3. Action-Selection Methodologies & Implementations

## 3.1 Action-Selection Methodologies

The term "action-selection" refers to the method, or methods, used by an agent to determine the next course of action it wishes to perform within its environment. Action-selection effectively combines an agents motor, perceptual, and cognitive schemas in a continuous resolution of "what to do next".

There are many possible ways which these courses of action can be selected. The most popular methods utilise probabilities, most commonly using some form of Bayesian theory, neuro-evolutionary or connectionist methods such as neural networks, reactive methodologies such as potential fields, and non-supervised learning techniques such as reinforcement learning or Q-learning.

These techniques are discussed in greater detail below.

### 3.1.1 Bayesian Theory

Bayesian theory is a useful tool in action-selection as it "supports the calculation of more complex probabilities from previously known results" (Luger, 2002), so through the accumulation of statistics on the possible world states, possible actions and their outcomes, an agent can begin to determine which actions are most likely to lead to optimal outcomes such as scoring a goal or prolonging an episode of keepaway.

These probabilities are calculated using prior and posterior, also called conditional, probabilities P(hypothesis) and P(hypothesis | evidence) forming Bayes theorem below:

$$P(H_a \mid E) = \frac{P(E \mid H_a) \times P(H_a)}{{}^{n}\Sigma_{b=1} P(E \mid H_b) \times P(H_b)}$$

Where $P(H_a \mid E)$ is the probability that H is true given evidence E, $P(H_a)$ is the probability that $H_a$ is true overall, $P(E \mid H_a)$ is the probability of observing evidence E where $H_a$ is true and n is the number of possible hypothesis (Luger, 2002).

Applying this theorem to a game of robot soccer could give you the following:

$$P(Outcome_x \mid Action_y) = \frac{P(Action_y \mid Outcome_x) \times P(Outcome_x)}{{}^{n}\Sigma_{z=1} P(Action_y \mid Outcome_z) \times P(Outcome_z)}$$

Where $P(Outcome_x \mid Action_y)$ is the probability that $Outcome_x$ was the result of taking $Action_y$, $P(Outcome_x)$ is the probability of $Outcome_x$ occurring in a game, $P(Action_y \mid Outcome_x)$ is the probability of $Outcome_x$ being reached as a result of taking $Action_y$ and n is the number of possible actions available.

There are two main constraints to using Bayesian theory, firstly, all of the relationships between the evidence and hypotheses sets, and the relationships between the individual pieces of evidence must be able to be quantified as probabilities. Secondly, all relationships between evidence and hypothesis, or $P(E \mid H_k)$, must be calculated.

Computationally, these constraints require the probability tables to be rebuilt when new relationships between hypotheses and evidence, or actions and outcomes, are discovered. For real-time action-selection, using full Bayesian techniques is unrealistic, as computing restraints mean that the processing time required will be well outside that required for real-time processing.

Therefore, in order to be implemented successfully for real-time applications, Pearl proposed a pruning technique, Bayesian Belief Networks (Pearl, 1988), which reduces the complexity of the system by focusing on a smaller set of more relevant evidence which has allowed Bayesian theory to be used successfully in a number of time critical applications including pattern recognition and classification applications such as voice and facial recognition, and is a popular technique for filtering email for spam and other forms of unsolicited or objectionable email.

### 3.1.2 Neural Networks

Neural networks are a biologically inspired machine learning technique. Also known as parallel distributed processing (PDP) or connectionist systems. In these systems, learning is considered to be the result of the constant adjustment of the interactions between simple neuron like components.

The basis of neural networks is the artificial neuron. This artificial neuron consists of a series of input signals, a set of weights, an activation level, and a threshold function. In addition to these individual neuron properties, the network of neurons also includes a network topology, a learning algorithm and an encoding scheme.

The neurons input signals come either from the environment or are passed to the neuron from the activation of other neurons. Inputs are typically discrete and tend to come from the set of real numbers. A set of weights are used to describe the strength of the connections between neurons. The activation level is calculated by adding the strength of each of the neurons input signals after each input has been scaled by its weight. Finally, the threshold function computes the neurons output value by determining how far the neurons activation level is above or below a determined threshold value.

As for the global properties, the network topology refers to the patterns formed by the connections between the individual neurons. This is the primary source of the networks inductive bias, basically the "role of the knowledge, expectations and tools the problem solver brings to problem solving" (Luger, 2002, p467). Finally, the encoding scheme represents the interpretation placed on the data by the network and the results of its processing. These global and local properties are illustrated in a simple neural network in figure 3.1 below:

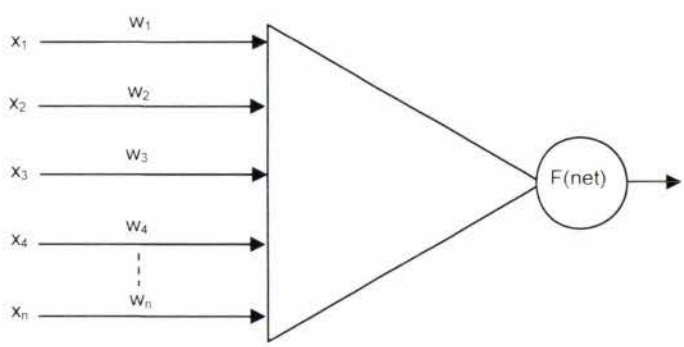



Figure 3.1: Artificial neuron with input vectors (x), weighted input lines (w) and a thresholding function f()that determines the neurons output value.

Neural networks are a popular method of action-selection in RoboCup. Implementations tend to use a number of smaller networks, usually one for each low level skill whose outputs combine to form an emergent behaviour.

One successful method was presented by Whiteson and Stone which proposed using a hierarchy of neural networks to train agents to use high level behaviours (Whiteson & Stone, 2003) such as `PassEvaluate()` and `GetOpen()`. Both of these high level behaviours require the knowledge of lower level skills including `Pass()` and `Intercept()`.Figure 3.2 below illustrates the topologies of the `PassEvaluate()` and `GetOpen()` behaviours:


Figure 3.2: Example network topologies for the PassEvaluate() and GetOpen() functions.

The inputs to these networks are the state variables used in the keepaway sub-task including the distance and angles to the other players and the ball and the output represents either the confidence in the action or the desired heading and speed. Whiteson and Stone's experiments proved that concurrent layered learning using multiple neural networks is an effective method of action-selection and was more effective than traditional layered learning and neural network approaches (Whiteson & Stone, 2003).

Although the concurrent learning technique cut down on the overall training time required for the individual networks, neural networks techniques are still considered to be heavily time intensive.

Overall, neural networks are a versatile method of selecting actions in complex domains. Although they often take a long period of time to train and plan as the researcher must consider the topology, a method of encoding all inputs and outputs, not to mention what the results might "mean" when they are processed, neural networks are still considered to be a preferable and more powerful method of action-selection.

As well as action-selection, neural networks work particularly well with symbolic problems such as classification, pattern recognition and noise filtering.

### 3.1.3 Potential Field Methodologies

Potential Fields is a popular reactive architecture most commonly used for motion planning. Vectors are used to represent possible motion or behaviour options and combinations of vectors are then summed to form an emergent behaviour (Murphy, 2000).

Any perceivable object in the robots world exerts a force on the surrounding space. The field represents what the robot should do if it comes in range of that field (Murphy, 2000). For example, a uniform field portrays a "go in this direction" behaviour, an attractive field represents a "go towards

goal" behaviour, a repulsive field represents an "avoid" behaviour and a tangential field is used to spin a robot around in an "avoid obstacle" behaviour these fields are illustrated in figure 3.3 below:

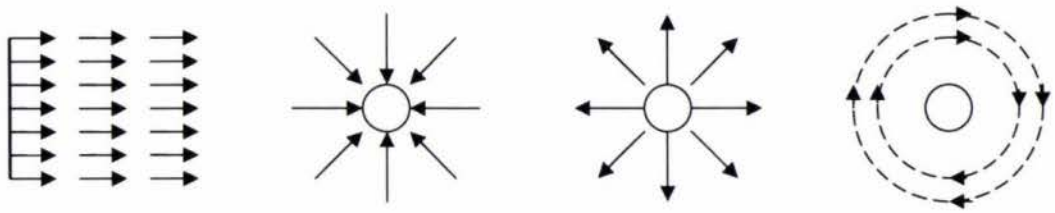


Figure 3.3: Uniform, attractive, repulsive and tangential potential fields.

Magnitude profiles are used to represent the way the magnitudes of the vectors change within the potential field. The three most common profiles are constant, linear drop-off and exponential drop-off. These profiles ensure that the robots response to the field stimuli is proportional to the strength of the field.

Potential fields are popular in RoboCup Soccer as they allow researchers to instil certain basic behaviours into players by repelling them from the boundary, attracting them to the opponent's goal and making them aware of the offside line. Also, as a reactive architecture, the use of potential fields for lower level behaviours allows researchers to build subsumption like behaviour architectures with combinations of planned actions and emergent behaviours.

### 3.1.4 Reinforcement Learning

Reinforcement learning, introduced by Sutton and Barto, is a form of machine learning most commonly used for problem solving (Sutton & Barto, 1998). "Reinforcement learning is all about sequential decision making, achieving delayed goals, and handling noise and stochasticity. It is also oriented toward making decisions rapidly rather than relying on extensive deliberation or meta-reasoning" (Stone et al, 2005a, p3).

Learning takes place through a process of trial and error when an agent begins to interact with its environment and explore it using randomly chosen actions (random policy). The agent's choice of actions is reinforced through the receipt of rewards when these actions lead to positive outcomes. After what is referred to as a training period, these action choices become optimised as the agent gains the ability to model the relationships between the state of the environment, and the sets of available actions and rewards.

A common policy used with reinforcement learning is an aptly named "greedy policy" where an agent uses its previous experience (or training) to take actions solely to better the received rewards,

however, this type of policy has it's draw backs as it relies on the rewards to be certain and well-defined (Sutton & Barto, 1998). The use of the greedy policy in an environment containing uncertainty, or which is not well defined, can cause the agent to form emergent behaviours away from what a human observer would consider optimal.

For example, in a game of soccer, a goal-keeping agent could begin to favour attacking actions over defensive actions leading to the opposition scoring many goals as the agent has learnt that the rewards for choosing attacking actions are better than for choosing defensive actions.

The conventional approach to solving this problem is to force the agent into choosing an occasional action at random. Another approach is to use a spin-off of reinforcement learning known as Q-learning, discussed in detail below, where the rewards received are discounted to represent uncertainty.

Reinforcement learning has been successfully used in backgammon, helicopter control, elevator control and keepaway soccer (Stone et al, 2005a)

Although, in principle, reinforcement learning methods are suitable for use in RoboCup, the simulated soccer environment presents many challenges to this type of action-selection policy because of it's large state space containing hidden and/or uncertain states as agents only have a partial view of their surroundings, multiple independent agents learning simultaneously and long or variable delays between when an action is taken and when it's effects can be seen (Stone et al, 2005a).

To overcome these problems, agents are frequently taught independently, but share reward values and the large state space is handled using function approximation although many forms of approximation don't converge to an optimal solution or are not well suited to learning complex functions. The best solution to date is the linear Sarsa ($\lambda$) method of function approximation (Sutton & Barto, 1998), also proposed by Sutton & Barto, which is considered to have a high convergence rate as long as the action-selection policy is continuous (Stone et al, 2005a).

Whether combining reinforcement learning with a function approximation method, such as linear Sarsa, is advantageous over using an alternative method such as Q-learning, depends greatly on the preference of researchers involved, the problems that they are trying to solve, or the theories they are trying to explore. However, one consensus is clear, that both methods are effective ways of dealing with action-selection in large state spaces and neither method has yet been tested fully enough to determine an optimal solution.

### 3.1.5 Q-Learning

Watkins introduced Q-learning, an alternative method of reinforcement learning (Watkins, 1989). In Q-learning, the agent is said to exist in a world that can be modelled as a Markov Decision Process (MDP) (Luger, 2002).

A MDP is used to "model an interactive system with an evolving state" (Strens, 2000, p1). It is defined by the quadruple (S, A, T, R) where S and A are sets of states and actions respectively, T is a probabilistic state-action transition function and R($s$, $a$) is the reward for the state-action pair ($s$, $a$). The main component in this quadruple is T, the transition function, which is used to generate a scalar reward from $R(s, a)$ where $X_t$ is the state of the system at time t, $X_{t+1}$ is the state at the next time step and $Y_t$ is the action taken on the system by the agent:

$$T(s, a, s') = P(X_{t+1} = s' \mid X_t = s, Y_t = a) \quad (1)$$

Therefore, in order to use Q-learning, an agent must be capable of observing discrete world states and execute discrete actions, and maintain a set of running estimates of $Q(s, a)$, (the quality function), in order to maximise a discounted return.

This discounted return is the sum of the rewards received for the duration of learning so that rewards received sooner are considered to be of greater significance. The quality function $Q(s, a)$ is defined as the discounted return the agent receives when action $a$ is taken in state $s$, assuming an optimal result occurred (Strens, 2000).

Assuming estimates of $Q(s, a)$ can be learnt, then the optimal choice in state $s$ is to choose an action with the largest Q-value.

Q-learning works by maintaining an array of running estimates that are updated at each time step. When action $a$ in state $s$ leads to state $s'$ with the instant reward $r$, the Q-learning rule updates the existing estimate for $Q(s,a)$ as follows:

$$\hat{Q}(s, a) \leftarrow (1 - \alpha)\, \hat{Q}(s, a) + \alpha(r + \gamma \max_{a'} \hat{Q}(s', a')) \quad (2)$$

This is a linear combination of the previously stored estimate with the estimate obtained from the most recent observation. The learning rate $\alpha$ determines the proportions in which the previous and current estimates are combined and serves to average over several forms of uncertainty including

the probabilistic nature of state transitions and immediate rewards and the error in the current estimates of $Q$ (Strens, 2000).

Q-learning, purely from the manipulation of Q-values, can be very effective if a large number of trials can be performed, but much faster learning can be obtained if Q-learning is combined with a Markov Decision Process.

In order to include the MDP in Q-learning, equation (1) and equation (2) are combined to form a more explicit expression of $Q(s, a)$ where E[] is the expectation operator:

$$Q(s, a) = E[R(s,a)] + \gamma \Sigma_{s'} T(s, a, s') \max_{a'} Q(s', a') \quad (3)$$

Like most learning methods, Q-learning is an effective way of conducting action-selection, but it also suffers from the same drawbacks such as the amount of training time required before optimal mappings are learnt and an instability in non-continuous state spaces.

## 3.2 Implementation

### 3.2.1 Simulation Environment

The experiments were conducted using the RoboCup Soccer Server with the additional Keepaway Framework infrastructure as described below. The simulation will be run on a single node of the Helix cluster at Massey University. This node will be running Red Hat Linux version 7.3 and contains dual AMD Athlon MP-2100 processors with 2GB DDRAM. Specific operational details can be found in the user guide located in section A1.1 of the appendix.

#### 3.2.1.1 RoboCup Soccer Server

The RoboCup Soccer Server is a physical soccer simulation system capable of visualising a game on a computer screen. The Soccer Server was written to support "competition among multiple virtual soccer players in an uncertain multi-agent environment, with real-time demands as well as semi-structured conditions" (Chen et al, 2003).

The RoboCup Soccer Server consists of two programs the Soccer Server, responsible for simulating the soccer game, and the Soccer Monitor, responsible for visualising the game.

The Soccer Server runs in a client server style with each team connecting up to eleven players (the clients) and one coach and one trainer to the Server. Each player runs as a separate process and receives visual and auditory sensor information and sends control instructions via a UDP/IP socket connected to a specified port. In a competition, this architecture allows the Server to run on a dedicated machine with the player clients connecting to it from a permitted number of separate machines. From a research perspective, this architecture also allows researchers to run the Soccer Server from a single machine, meaning it doesn't require hefty computing resources.

The Soccer Server runs as a real time system simulating discrete time intervals or cycles. Every 100ms cycle, players can request information on the current server state and can instigate a number of actions including dash, kick, move, and turn which are actioned during the transition between cycles. If a number of players have requested actions on the same object, for example they have all kicked the ball, the resultant vector and velocity (up to a maximum value) for the ball is calculated then applied.

After applying all of the actions in a randomised order, the server then checks to ensure no collisions have taken place and if there was a collision, updates the velocities and vectors appropriately for the objects involved.

The virtual referee then checks to ensure that the new simulation state doesn't require a change of play mode, for example, a reset after a goal is scored. If a change of play mode is required, the referee sends a message to all players immediately. Finally, the player's stamina is updated to reflect the actions taken.

The Soccer Monitor allows the action taking place on the server to be visualised shows the score, team names, positions of players on field, and the ball. It also provides a simple interface to the server which allows users to interact with the server, such as signalling the kick-off or start of the game. The Soccer Monitor also allows the user to zoom into areas of the field, gives current positions and velocities of the players and the ball as well as their view cones and stamina. In the case of a heterogeneous team, the Soccer Monitor can also display the various player types.

The game is governed by both a human referee, connected via the Soccer Monitor who polices certain aspects of game play that cannot be easily governed by the virtual referee this includes surrounding the ball, blocking the goal with too many players, not putting the ball into play after an given number of cycles, intentionally blocking movement, abusing the goalie catch command, flooding the server with messages, and other inappropriate behaviour.

As the Soccer Server was developed with an eye towards realistic simulation, a number of models are used to ensure that the simulated players reflect the sensor capabilities of human players these models include the Aural Sensor Model, the Visual Sensor Model and the Stamina Model.

The Aural Sensor Model allows players to detect messages sent by the referee, the coaches and other players on the field. To avoid one team from taking advantage of the aural message system and overloading the system, making the other team's communication useless, all players have separate hearing capacities. These capacities are diminished with each message received and are slowly built back up with each passing message free cycle. This means that a player can hear one message at most every second cycle. Audio is also constrained by distance, so only players inside the audio distance with the required amount of hearing capacity, can hear a message transmitted by another player.

The Vision Sensor Model reports on objects currently within the player's field of view. This field of view is dependant on a number of factors as seen in figure 3.4 below. This information, once gathered, is automatically sent to the player every sense step currently 150 ms.

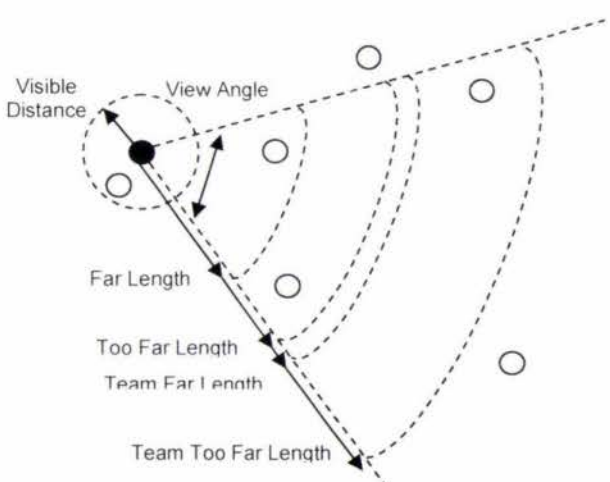



Figure 3.4: Model of a player's visual sensor showing its view cone, its visible distance where any agents seen can be identified with a confidence of 100%. The confidence in the visual data decreases by given proportions as players move farther outside the visible distance into the marked zones.

### 3.2.1.2 Keepaway Soccer Framework

As discussed in the previous chapter, keepaway soccer is a sub-problem of the RoboCup simulated soccer domain in which one team, the keepers try to maintain possession of the ball within a

bounded area while the opposing team, the takers, attempts to gain possession. An episode ends when either the takers gain possession of the ball or the ball goes out of bounds.
Game play is controlled by an omniscient coach, which determines when an episode ends and resets the players accordingly.

The keepers team begins each episode with possession of the ball given to the keeper positioned at the top left hand corner of the region. With each episode start, the keepers are randomly assigned to four starting positions: one keeper to the top left corner, with the ball, one to the top and bottom right corners, and any remaining keepers to the centre of the region. The takers always start in the bottom left corner. These positions are illustrated in figure 3.5 below:


Figure 3.5: Starting positions for 3 vs.2 keepaway soccer.

As keepaway is implemented on the RoboCup simulator, keepaway players are provided with visual data every 150ms and can execute primitive commands such as turn, kick or dash every 100ms. Inter-agent communication is prohibited except for verbal messages communicated through the simulator server. Also, as every player runs as a separate process, players have the ability to learn independently of each other and create their own control policies.

Keepaway has a number of parameters which can be modified, such as the size of the region, the number of keepers and the number of takers.
The takers follow a hand-coded action-selection policy stating that the closest or fastest taker to the ball should attempt to gain possession of the ball, otherwise, a taker should mark any open keepers.

Keepers on the other hand are only given a choice of actions when they are in possession of the ball. This is where the action-selection policies come into play. The player with the ball, receives a world state, made up of the series of variables detailed below and is able to chose from a series of actions numbered from 0 to the number of keepers – 1 where 0 represents the `HoldBall()` action and the remaining numbers represent `PassToKeeper(n)`.

The world state in a 3 vs. 2 keepaway game is represented by the following variables:

- The distances from each keeper to the centre.
- The distances from each taker to the centre.
- The distances from the current keeper (K1) to the remaining keepers.
- The distances from K1 to each taker.
- The distance between the closest taker to the next keeper, K2.
- The distance between the closest taker to K3.
- The angle between K2 and the closest taker.
- The angle between K3 and the closest taker.

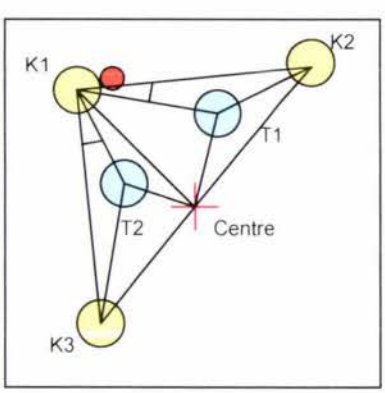



Figure 3.6: Representation of keepaway state variables.

Included with the Keepaway Framework, are a set of three static keepaway policies developed by Stone, Kuhlmann, Taylor and Liu (Stone et al, 2005). The 'Always Hold' policy dictates that once a keeper has possession of the ball it maintains sole possession of it and does not pass. The 'Random' policy sees the keepers randomly select one of the available actions. Thirdly, the 'Hand-Coded' policy is a simple strategy that compels the keeper to hold the ball when no takers are within a given distance otherwise, they determine their most open team mate and pass to them.

These three policies will be used as experimental controls with which to compare the results of the proposed confidence model too, as discussed later in section 3.2.2.1.

### 3.2.2 Action-Selection Implementations

So far, the action-selection approaches discussed have been action-centric, i.e. the action is chosen on its own merits and the player or players carrying out the action are of secondary concern.

For example, in a heterogeneous human soccer team each player has an individual set of abilities. These abilities represent their level of proficiency for certain skills including passing, dribbling, marking, evading, goal scoring etc. When a human player makes a decision, they include information gathered, not only from their environment, but also from their observations and faith in their team-mates skills.

An example of this behaviour is illustrated in figure 3.7 below. Say player A is in possession of the ball and has a choice of whether to pass to player B, player C or player D. Players B and C are closer, but both are being loosely marked by opponents. Player D is currently un-marked, but a successful pass to player D would require the ball to pass both opponents.

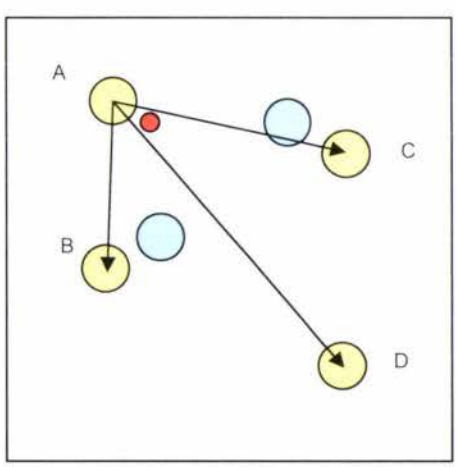



Figure 3.7: Human soccer player decision scenario.

Player A therefore has a number of action choices, hold onto the ball for a period of time, then re-evaluate the situation, or pass to players B, C or D.

In all likelihood, player A will evaluate these action choices using a heuristic approach, based on his or her knowledge of the game and his/her evaluations of the other players. These evaluations would have been created from observations of the players and lead to the player forming a model of their respective playing abilities. Therefore when player A is deciding on which player to pass to,

he/she not only takes proximity into account, but also the likelihoods of opponents intercepting the ball mid-pass and team-mates being able to receive the pass successfully. In other words, the player determines which team-mate they have the greatest confidence in, in order to make the decision to pass to them.

Now in the above scenario, although passing to player D seems an unlikely choice, player A might have more confidence in player D successfully receiving the pass and more confidence in players B and C successfully evading, then marking their opponents over B or C successfully evading the opponents then receiving the pass.

I propose a method of attaching this player ability confidence onto an action-selection policy so that decisions made by an agent are influenced not only on its past experiences in the environment, but also by its past experiences with its team-mates and opponents.

#### 3.2.2.1 Confidence Model

In this model, the confidence levels represent a percentage of how certain a player is in the abilities of its peers and its opponents. Utilising principles of Bayes theories, the confidence levels will act as a posterior probability of previous skill performances providing a gauge of how similar actions taken in similar future scenarios will turn out. These values will be constantly updated just as with a learning policy.

In order for the individual strengths and weaknesses of players in the RoboCup soccer domain to be quantified, I propose that the following statistics be used to represent four of the basic soccer skills:

- Passing: The Passing statistic refers to the current player's ability to successfully pass to another player.
- Intercepting: The Intercepting statistic measures the player's ability to successfully receive a pass from a teammate or intercept an opposition pass.
- Blocking: The Blocking statistic measures a player's ability to mark an opposition player so they are not open to receive a pass.
- Evading: The Evading statistic measures a player's ability to escape being marked by another player.

However, as the keepaway domain is only a small part of the RoboCup domain, and it is envisaged that not all statistics will be fully utilised at this time.

These statistics can then be combined to form a total confidence level for each proposed action, for example, going back to the player A, B, C and D scenario of the previous section, if P, I, B, and E represent the above confidence statistics, and $K_A$, $K_B$, $K_C$, $K_D$ are considered the four players in their role as keepers and $T_1$ and $T_2$ represent their opposing takers, then the following equations represent the pass options to each player and $K_A$ holding the ball:

$$PassBall(K_A, K_B) = ((P_{KA} + I_{KB}) - I_{T1}) + (E_{KB} - B_{T1})$$

$$PassBall(K_A, K_C) = ((P_{KA} + I_{KC}) - I_{T2}) + (E_{KC} - B_{T2})$$

$$PassBall(K_A, K_D) = ((P_{KA} + I_{KD}) - (I_{T1} + I_{T2})) + ((E_{KB} - B_{T1}) + (E_{KC} - B_{T2})) + ((B_{KB} - E_{T1}) + (M_{KC} - E_{T2}))$$

$$HoldBall(K_A) = E_{KA} - ((E_{T1} - B_{KB}) + (E_{T2} - B_{KC}))$$

As demonstrated, these equations are made up of a series of paired values as it is assumed that opposition players will in turn react in an appropriate way such as trying to intercept a pass or mark an open player. These paired values can then be easily chained to form more complex equations representing more complex scenarios.

These statistics will be maintained in a table which will record the total number of times each skill was used by a player and the number of times this action lead to a positive outcome. The percentage of positive outcomes is then calculated and forms the player's confidence level for this skill.

As the combination of the confidence level statistics depends on the current game state as well as the possible actions available, the next step in forming a functioning confidence level based action-selection policy is finding a suitable technique for implementation.

#### 3.2.2.2 Direct Implementation

The first method to be implemented will be a direct use of the confidence level model to determine actions. This direct use will be in a similar form to the equations discussed in the previous section, but once the answer to each calculation has been computed, the resultant action will be the one with the highest confidence level. Figure 3.8 below illustrates a suggested program flow of such an approach:

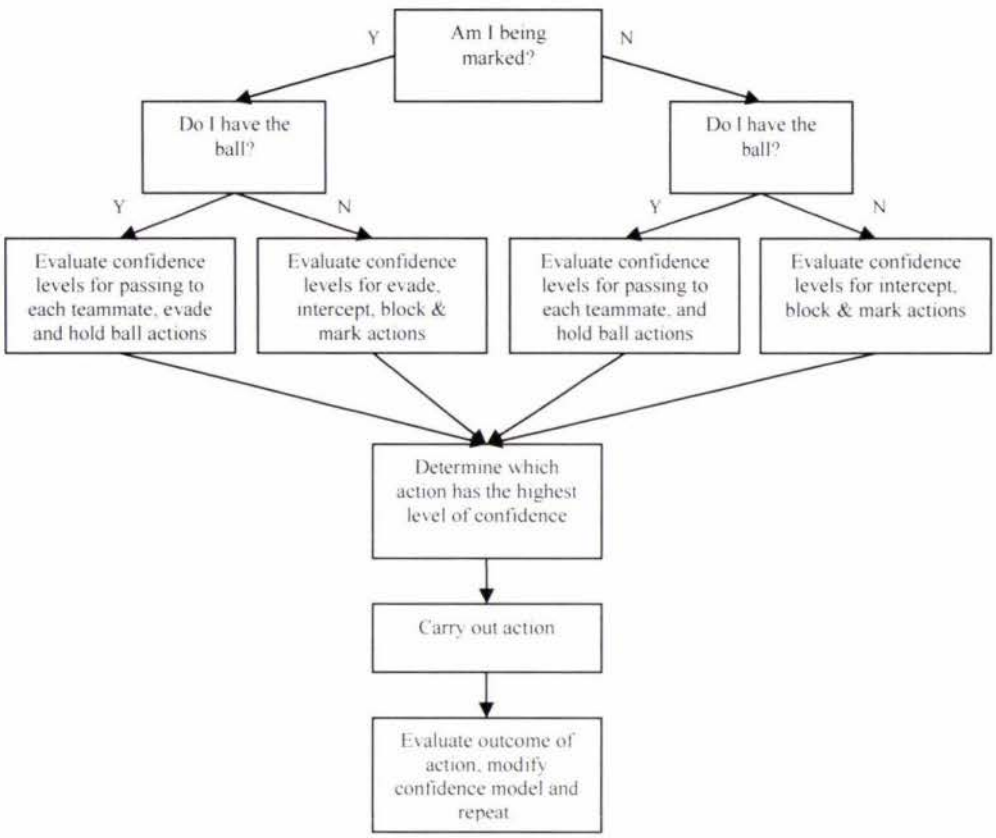



Figure 3.8: A direct confidence level approach to action-selection.

Initial empirical testing pointed out two potential problems with this proposed implementation. Firstly, this method suffered from the "local minima" problem discussed in section 3.1.4, when a confidence level for certain action becomes so high that no other actions are considered. This was solved by adding in an 'every so often, choose random action' step.

Secondly, in the early stages of model building, the direct implementation often failed to handle situations where a keeper with a high intercept confidence is directly behind one or more takers with poor intercept confidences. This, more often then not, lead the keeper with the ball to pass directly into the path of a taker, who then gained possession of the ball, causing the episode to end. As a solution, I proposed a further implementation combining potential fields with the confidence model, so that this behaviour is discouraged.

### 3.2.2.3 Potential Field Confidence Model Implementation

The next approach to be tested will involve the combination of a popular reactive approach, potential fields, and the proposed confidence level model. This implementation effectively adds the ability to further distinguish optimal actions by applying a level of influence, or a limit to how far the confidences are "felt" by the other players.

This is achieved by surrounding each player with a potential field with a radius equal to the combination of appropriate confidence statistics multiplied by the distance between the players. In the case of a keeper, the field would be determined using the pass and intercept statistics and for a taker only the intercept statistic is included. For example, if the target keeper has an intercepting confidence of 25% and the keeper with the ball has a passing confidence of 25% and they are separated by a distance of 10 metres, then the radius of the target keepers confidence field is 5 metres.

To account for the other portion of the confidence model equations, the adjustment based on the competency of the takers, fields are accordingly modified so that no keeper's field overlaps with a takers. This is achieved by determining the area of the overlap, if any, and then subtracting this overlap from the area of the effected keeper's potential field.

This is further illustrated in the figure below which displays an example of the problem observed in the direct implementation where the passing route to the keeper with the highest resultant confidence level is non-optimal as it is being blocked by one or more takers.

In the early stages of the direct implementation, this pass action was initially considered optimal until the model has a chance to mature, often requiring many thousands of episodes training which defeated one of the goals of the implementation, to provide a method of action-selection that wasn't training intensive.

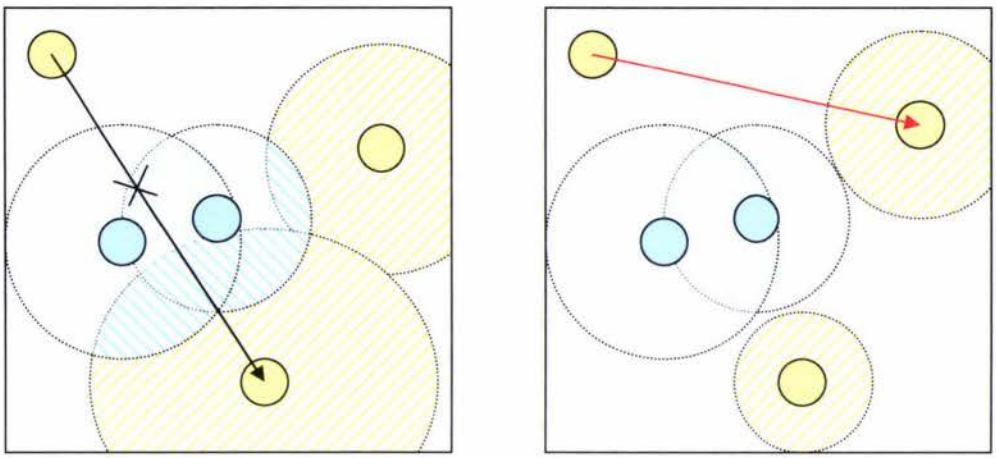


Figure 3.9: A non-optimal passing scenario is displayed to the left with a visualisation of the beginning stages of the potential fields hybrid solution. The results of the solution, once optimised, are shown to the right along with the emergent behaviour.

The first portion of figure 3.9 illustrates the non-optimal passing behaviour favoured by a juvenile direct confidence implementation and the placement of the non-optimised potential fields for the hybrid implementation. The second portion of the figure shows the emergent behaviour of the system once the fields have been pruned to account for any overlap.

## 3.3 Empirical Testing Regime

To demonstrate the generality of the approach, it will be applied to a number of keepaway task variations including fielding different numbers of keepers and takers over different field sizes.

### 3.3.1 Benchmark Testing

In order to have comparisons to test the feasibility of the confidence model against, the first round of testing will involve re-benchmarking the three policies provided with the Keepaway Framework by Stone et al, 'Always Hold', 'Random' and 'Hand Coded'.

This re-testing is necessary as the test runs for the provided Framework only spanned 1,000 episodes per policy and it was important to ensure there would be no inconsistencies with the data collected from testing the confidence model.

Benchmark testing will involve the same test setups as described below for feasibility testing, but will involve only the 'Always Hold', 'Random' and 'Hand Coded' policies.

Results collected from these tests will be summarised and compared with the provided results to ensure that the simulation environment is running consistently for the remainder of the testing.

### 3.3.2 Feasibility Testing

The main feasibility testing will take place over 10,000 episodes per policy, per test setup as detailed below:

| Number of Keepers | Number of Takers | Field Size |
|---|---|---|
| 2 | 1 | 15m x 15m |
| 3 | 2 | 20m x 20m |
| 4 | 3 | 25m x 25m |
| 5 | 4 | 30m x 30m |

Table 3.1: Feasibility testing regime.

These tests will be carried out for each of the confidence model policies 'Direct' and 'Potential Fields' and compared with the Framework results collected during benchmark testing phase.

The results will also be summarised and individual policy performances compared to determine if the confidence model approaches are feasible action-selection implementations.

### 3.3.3 Scalability Testing

Once the feasibility of the confidence model approaches have been ascertained, additional testing will then be conducted to demonstrate the scalability and generality of the confidence approach.

These tests will include altering the prescribed field sizes for 3 vs. 2 games to 40m x 40m and 15m x 15m and scaling testing up to 11 vs. 11 on a full field (105m x 68m).

These additional tests will be conducted over a run of 1000 episodes per policy.

### 3.3.4 Entropy Testing

The final round of testing will aim to put an end to the confusion surrounding the entropy of the teams.

The first round of entropy testing will involve a standard 3 vs. 2 team running the 'Direct' policy playing 1000 episodes on a 20m x 20m field.

The confidence levels observed by each keeper will then be averaged to give an overall performance indicator for each player.

The second round of entropy testing will involve the same 3 vs. 2 setup as the first round with one difference, each keeper will be 'handicapped' by 'sleep' loop which will countdown from a number generated randomly between 0 and a large seed number before the agent is able to complete it's action-selection.

This 'sleep' loop is designed to emulate the 'sleep' function common of a multi-threaded environment which enables a given thread to be paused or suspended for a certain amount of time before resuming its processing.

The results collected from these tests will then be compared and contrasted in order to determine if either team is performing heterogeneously or homogeneously.

## 3.4 Summary

Action-selection is a method that effectively combines an agents motor, perceptual, and cognitive schemas in a continuous resolution of "what do I do next?".

Many popular methodologies for action-selection were discussed in this chapter including Bayesian Theory, Neural Networks, Potential Fields, Reinforcement Learning, and Q-Learning and a trend towards action-centric action-selection methods was identified. This action-centricity implies that the action is chosen on its own merits and the player or players carrying out the action are secondary to concern. This in turn lead to the proposition of a statistics based player-centric approach which aimed to address this lack of player consideration in the action-selection process.

The core of this process is the use of confidence level statistics to represent the strengths and weaknesses of the individual players. For the keepaway domain which will be used as the test bed for this method, four statistics will be used: Passing, Intercepting, Marking and Evading.

The implementation of the Keepaway Framework and the RoboCup Soccer Server which supports it were reviewed in section 3.2.1 above. Version 0.4 of the Keepaway Framework was subsequently installed in conjunction with version 9.4.5 of the RoboCup Soccer Base and Server and version 9.3.7 of the RoboCup Soccer Server Monitor, the most stable releases available at the time. Most, however, have since been superseded by newer stable and beta releases.

Two methods were proposed to implement these confidence level statistics, the first a direct implementation and the second is a hybrid of the direct method and a potential fields method which was introduced to solve an optimum passing problem.

The final part of this chapter covered the testing regime for comparing the performance of the two proposed confidence level policies, 'Direct' and 'Potential Fields' against the policies provided by the Keepaway Framework, 'Always Hold', 'Random' and 'Hand Coded'.

Testing will take place in four phases, the initial phase of benchmark testing will aim to ascertain the stability of the Keepaway/Soccer Server implementation through comparisons of summaries of the results collected with the results provided. The second phase of testing will test the feasibility of the confidence model policies by comparing the results of the various test configurations collected in phase one with the results of testing the confidence model approaches for the same test configurations. Phase three will test the scalability of the confidence approaches by pitting them against the Framework policies in a series of extended tests. Finally, phase four will examine the entropy of the confidence model approach by conducting tests to determine if the observed player performance is heterogeneous or homogeneous.

The results of this testing can be found in the next chapter.

# 4. Results

## 4.1 Benchmark Testing

In order to have comparisons to test the feasibility of the confidence model against, this round of testing involved re-benchmarking the three policies provided with the Keepaway Framework (Stone et al, 2005), 'Always Hold', 'Random' and 'Hand Coded'.

The Framework provided the following set of mean and standard deviation statistics (given in seconds) as their benchmark of results conducted over 1000 episodes for teams of 3 vs. 2, 4 vs. 3 and 5 vs. 4 on field sizes of 20 metres x 20 metres (m), 25m x 25m and 30m x 30m respectively.

| Policy | 3 vs. 2 | | 4 vs. 3 | | 5 vs. 4 | |
|---|---|---|---|---|---|---|
| | Mean | Std Dev | Mean | Std Dev | Mean | Std Dev |
| **Always Hold** | 3.4 | 1.5 | 4.1 | 1.8 | 4.8 | 2.2 |
| **Random** | 7.5 | 3.7 | 8.3 | 4.4 | 9.5 | 5.1 |
| **Hand Coded** | 8.3 | 4.7 | 9.2 | 5.2 | 10.8 | 6.7 |

Table 4.1: Keepaway Framework results provided by Stone et al (2005).

The following results were obtained from conducting the same tests over 10,000 episodes using a single node on the Helix cluster at Massey University.

| Policy | 3 vs. 2 | | 4 vs. 3 | | 5 vs. 4 | |
|---|---|---|---|---|---|---|
| | Mean | Std Dev | Mean | Std Dev | Mean | Std Dev |
| **Always Hold** | 3.6 | 1.6 | 4.3 | 1.9 | 4.7 | 2.0 |
| **Random** | 8.2 | 4.3 | 8.4 | 4.3 | 9.5 | 4.7 |
| **Hand Coded** | 7.9 | 4.3 | 8.9 | 5.1 | 10.0 | 5.9 |

Table 4.2: Results recorded from testing conducted over 10,000 episodes on the Helix computer system.

These results, on the whole, proved fairly consistent as can be seen from the following table which demonstrates the difference calculated between the provided benchmark results in table 4.1 and the actual results obtained from Helix in table 4.2. These small variations in performance were expected due to the potential differences in the computer hardware used and the increased number of episodes recorded.

| Policy | 3 vs. 2 | | 4 vs. 3 | | 5 vs. 4 | |
|---|---|---|---|---|---|---|
| | Mean | Std Dev | Mean | Std Dev | Mean | Std Dev |
| **Always Hold** | 0.17 | 0.13 | 0.17 | 0.10 | -0.07 | -0.23 |
| **Random** | 0.74 | 0.55 | 0.11 | -0.09 | 0.00 | -0.41 |
| **Hand Coded** | -0.41 | -0.37 | -0.25 | -0.11 | -0.76 | -0.83 |

Table 4.3: Observed difference between results benchmarked by Stone et al and collected from Helix.

However, in order to achieve this small variation, many test runs had to be repeated as the outcomes produced early on were clearly outliers from the provided Framework as some means and standard deviations were between +/- 4.0 seconds and +/- 16.0 seconds above or below the given values.

After further testing this was found to be the result of an unusually heavy workload on the Helix node used and an error in the keepaway configuration settings that was being used to run the simulation system at twice real-time speed. By avoiding testing the bulk of the data during high use periods and running the simulation system at real-time, it is hoped that the system will perform with more stability.

Due to this initial system instability, the need to continue with the 10,000 episode testing regime is justified to even out any further kinks in the data although the commitment to test 10,000 episodes per policy requires a significant 10-fold increase in estimated testing time to over 400 hours total.

## 4.2 Feasibility Testing

In order to test the feasibility of the confidence model approach for the keepaway sub-domain of RoboCup simulated soccer, testing involved fielding varying numbers of players over varying field sizes for 10,000 episodes per test. The results of this testing are detailed below with full page graphs available in section A2.1 of the Appendix.

All policies will be referred to by their short titles, 'Always Hold', 'Random' and 'Hand Coded' for the policies provided in the Keepaway Framework and 'Direct' and 'Potential Fields' for the confidence model policies.

### 4.2.1 2 vs. 1 Keepaway

2 vs. 1 keepaway is played on a 15m by 15m field with one keeper positioned in each top corner and the single taker positioned at the bottom left corner.

Episode durations ranged from 3.0 to 5.5 seconds, with the heaviest concentration of durations between 4.0 and 5.0 seconds, as illustrated in the policy performance graph below. In order to graph all 10,000 results for each policy, each 100 episodes were averaged and those averages are shown in the figure below.

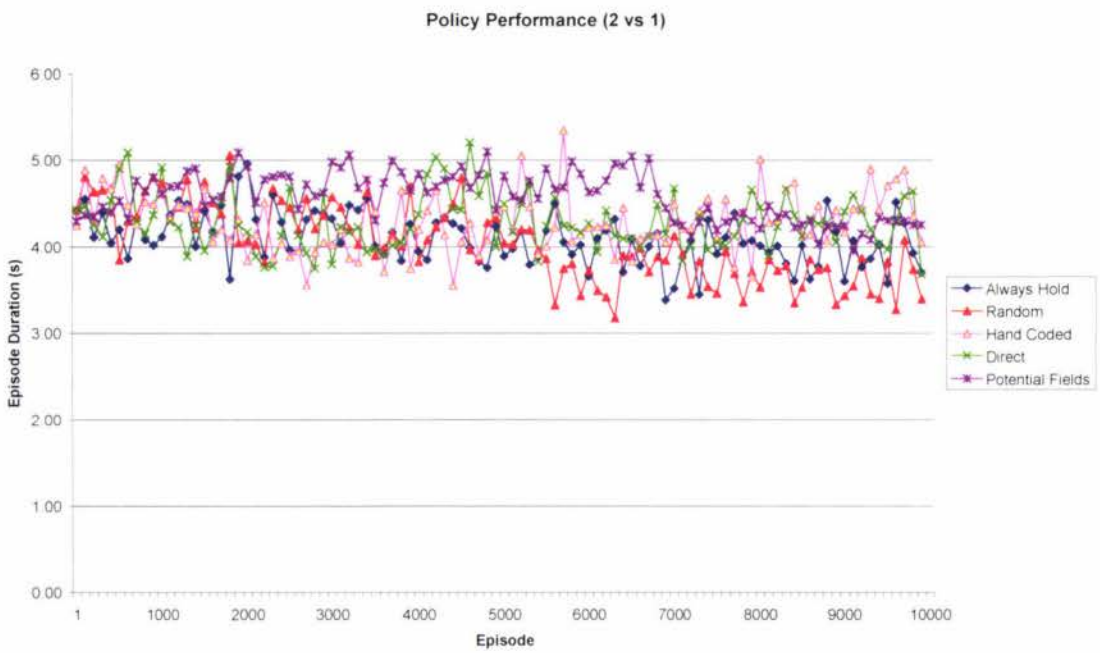



Figure 4.1: Policy Performance Graph for 2 vs. 1 keepway.

As shown, episode performances ranged chaotically, with no one policy clearly distinguishing itself. Further analysis shows that the distributions of the policies are fairly consistent as medians, ranges and highest and lowest values are very similar and all distributions are skewed to the left. These statistics have been graphed in the following box and whisker diagram.

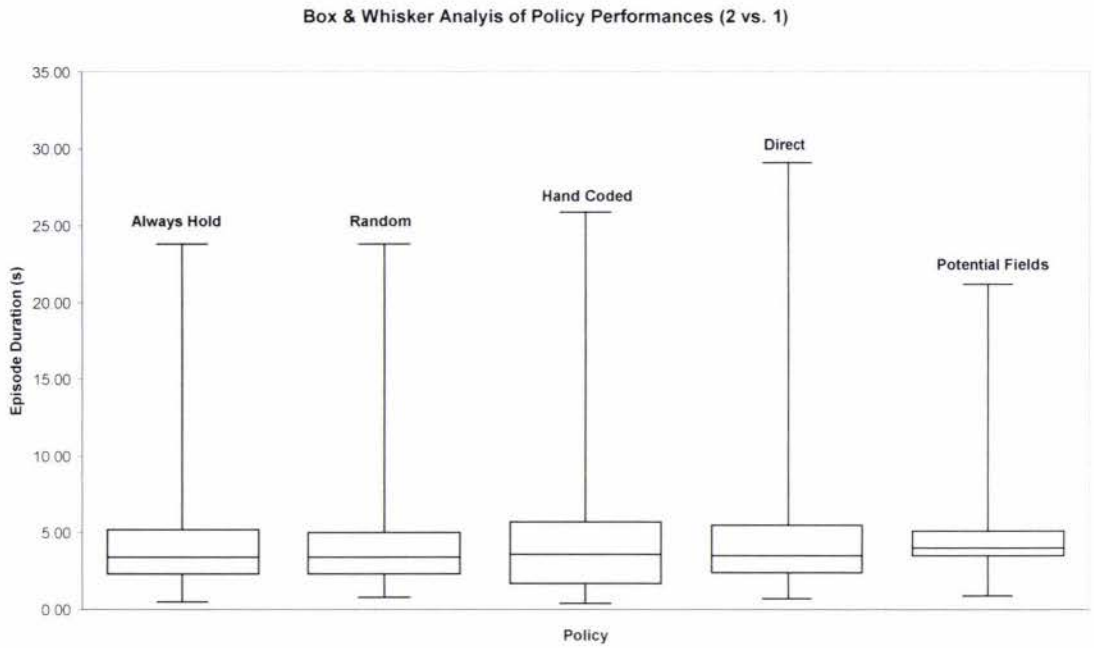



Figure 4.2: Box and whisker analysis of 2 vs. 1 policy performance.

As shown, the "Potential Fields' policy has the smallest distribution and the 'Direct' policy implementation, the largest.

Another set of interesting statistics collected were the episode outcomes. Although when an episode ends, no outcome is considered 'optimum', with a view to scaling the policy to a full 11 vs. 11 soccer implementation, an 'out' outcome could be considered better than a 'taken' outcome as an 'out' generally refers to a scenario of an ill aimed pass causing the ball to leave the boundary area. In a full game it could be construed that the team would be able to regain possession of the ball quickly in this event.

The episode outcomes for 2 vs. 1 were also fairly close with all policies experiencing at least a 100:900 ratio of 'takens' to 'outs'. The 'Potential Fields' hybrid policy performed with the least number of 'takens' at approximately 10:990 and the 'Hand Coded' policy achieved the most at approximately 60:940. This is illustrated further in the graph below:

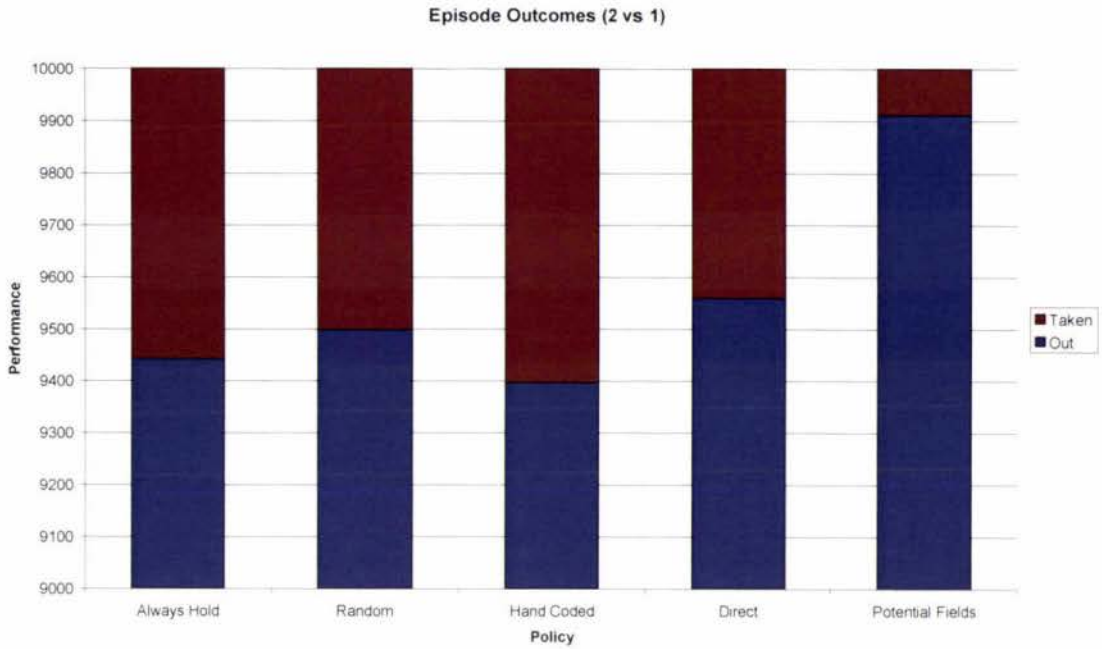



Figure 4.3: Graph depicting ratio of episode outcomes for 2 vs. 1 keepaway.

The overall performance of the action-selection policies can be seen more clearly in table 4.4 below. The 'Random' policy was the poorest performer averaging an episode duration of only 4.0 seconds and the 'Potential Fields' policy was the strongest performer with a slightly higher average duration of 4.6 seconds.

| Policy | Mean | Standard Deviation |
|---|---|---|
| **Always Hold** | 4.1 | 2.6 |
| **Random** | 4.0 | 2.6 |
| **Hand Coded** | 4.3 | 3.1 |
| **Direct** | 4.3 | 2.6 |
| **Potential Fields** | 4.6 | 1.6 |

Table 4.4: Summary of results for 2 vs. 1 keepaway soccer.

As previously illustrated in the performance graph featured in figure 4.2 and revealed in the above table, the 'Potential Fields' policy has the smallest standard deviation of all of the policies of 1.6 seconds. Incidentally, the 'Direct' and 'Hand Coded' policies performed equally, on average, however, the 'Direct' policy had a significantly smaller standard deviation of 2.6 seconds.

The 'Potential Fields' policy proved itself to be the strongest performer for the 2 vs. 1 test run, although the performance distributions had little between them. The overall performance of the 'Direct' method saw it achieve results slightly ahead of the 'Hand Coded' policy and its standard deviation was considerably lower, meaning its results were more consistent.

For the three provided policies, 'Always Hold', 'Random' and 'Hand Coded', the 'Hand Coded' policy proved to be the best performer, followed by the 'Always Hold' policy then the 'Random' policy. However, although this specific field size and team composition test was not part of the Framework, this result appears to be outside the norm where the 'Always Hold' policy has typically performed the worst.

Overall, for the first test run, both of the confidence model approaches have performed slightly better than their provided counterparts.

### 4.2.2 3 vs. 2 Keepaway

3 vs. 2 keepaway takes place on a 20m by 20m field with one keeper positioned in both the top corners and one keeper in the bottom right corner. Both takers are positioned in the bottom left hand corner.

Episode durations in this test run ranged from 3.0 seconds to 10.0 seconds with the performance of most policies falling between 7.0 and 9.0 seconds. The durations are illustrated in the following graph.

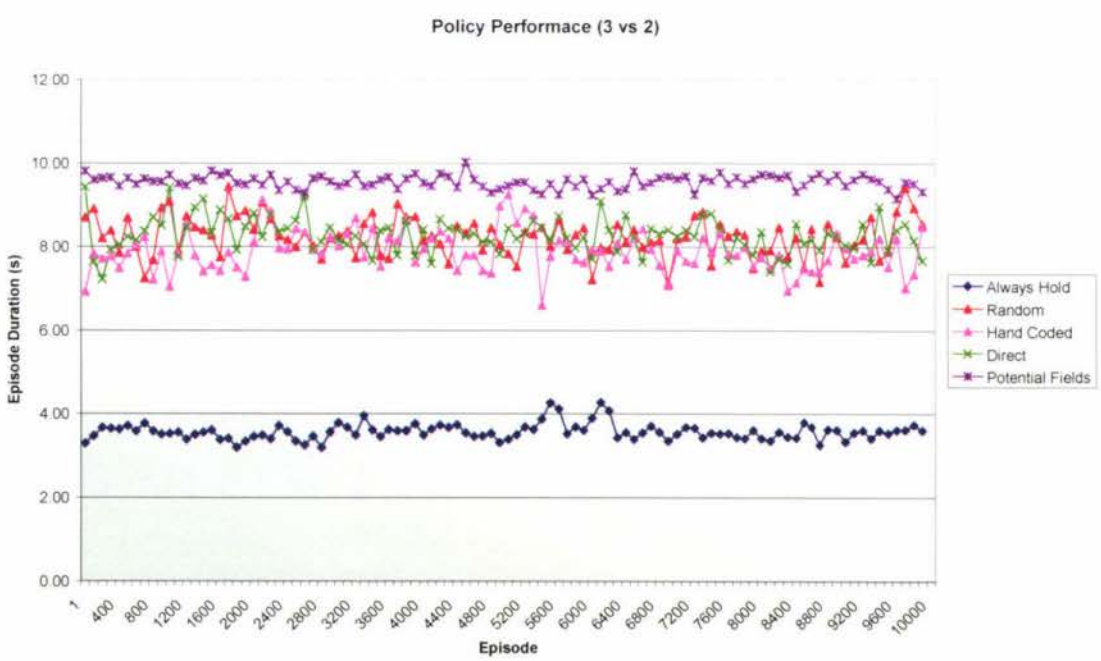



Figure 4.4: Policy performances for 3 vs. 2 keepaway.

As demonstrated, the 'Always Hold' and the 'Potential Fields' policies clearly distinguished themselves from the remaining three policies clustered around the 8.00 second mark. Although the clustered policies still appear to have heavily deviated performances, both the 'Always Hold' and 'Potential Fields' policies were more evenly distributed.

This distribution pattern can be seen more clearly in the following box and whisker diagrams where the 'Potential Field' policy displays the smallest distribution, followed by 'Always Hold' policy, the 'Random' policy, the 'Hand Coded' policy and finally the 'Direct' policy which shows the largest distribution.

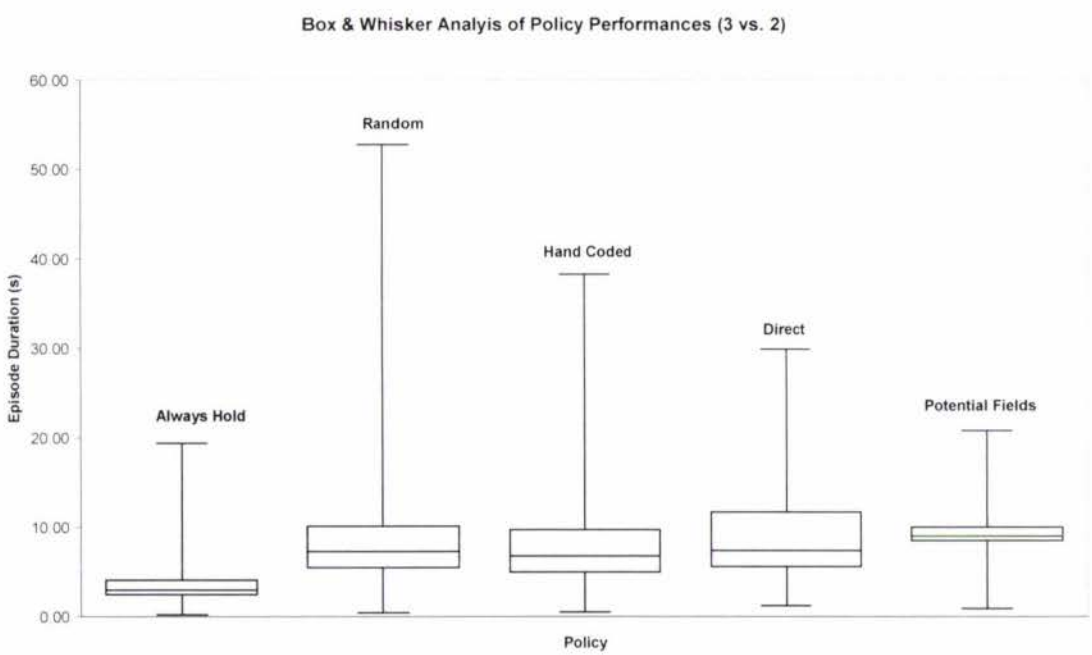



Figure 4.5: Box and whisker analysis of 3 vs. 2 policy performance.

Most of the above distributions proved consistent with the results for 2 vs. 1 with the exception of the 'Direct' distribution which was less heavily skewed, and the 'Potential Fields' distribution which was nearly symmetrical. The medians and inter-quartile ranges were also much more widely spread than the results observed from the 2 vs. 1 tests. There is also a clear distinction between each distribution unlike the 2 vs. 1 results which were all spaced very close together.

The episode outcomes, displayed in the following figure, also exhibit a similar trend to the 2 vs. 1 results, with the majority of episodes resulting in an 'out' as opposed to a 'taken' outcome with an average policy performance ratio of approximately 150:850 'takens' to 'outs'.

Once again, the 'Potential Fields' method performed with the least amount of 'takens' with a ratio of approximately 50:950 and the 'Hand Coded' method had the highest amount with a ratio of approximately 230:770 'takens' to 'outs'. These are shown in the figure below:

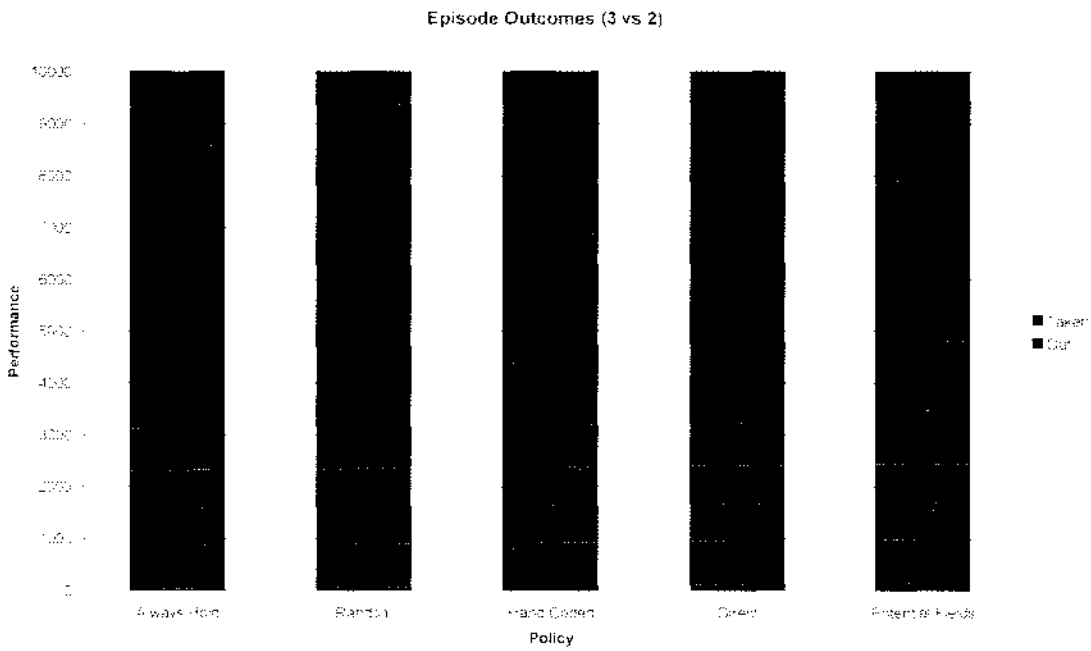



Figure 4.6: Graph depicting the ratio of outcomes of 3 vs. 2 keepaway

The overall performance of the action-selection policies has been summarised in table 4.5 below.

| Policy | Mean | Standard Deviation |
|---|---|---|
| **Always Hold** | 3.6 | 1.6 |
| **Random** | 8.2 | 4.3 |
| **Hand Coded** | 7.9 | 4.3 |
| **Direct** | 8.9 | 2.8 |
| **Potential Fields** | 9.6 | 1.6 |

Table 4.5: Summary of results for 3 vs. 2 keepaway soccer.

The 'Always Hold' policy was clearly the poorest performer averaging an episode duration of only 3.6 seconds, which was 0.5 seconds down from it's 2 vs. 1 performance, and the 'Potential Fields' policy was the strongest performer with an average duration of 9.6 seconds, 5.0 seconds better

than it's previous performance. Both of these policies shared the smallest standard deviation of 1.6 seconds.

The 'Direct' policy performed a close second with an average episode duration of 8.9 seconds, but maintained a smaller variation of 2.8 seconds which may explain its 0.7 second performance lead over the 'Random' policy, which not only shared the largest result deviation, but was also the best performer from the Framework policies by an average of 0.3 seconds.

As the first test configuration included in the Keepaway Framework, the provided results for 3 vs. 2 begin to demonstrate the strong performance trend shown throughout led by the 'Hand Coded' policy, then followed by the 'Random' and 'Always Hold' policies. However, as was the case with the 2 vs. 1 testing of the previous section, doubt has again been cast on this policy order as the 'Random' policy had clearly out performed the 'Hand Coded' policy by a small but significant 0.3 seconds.

Consistent with the results collected for 2 vs. 1 keepaway, the 'Potential Fields' policy again proved to be the best overall performer by a margin of 0.7 seconds.

### 4.2.3 4 vs. 3 Keepaway

4 vs. 3 keepaway is played on a 25m by 25m playing field with one keeper at each top corner, the bottom right hand corner and the remaining keeper positioned in the middle of the field with the takers positioned in the bottom left hand corner.

The policy performances for 4 vs. 3 ranged from approximately 4.0 seconds to 11.0 seconds with most policies durations falling between 8.0 and 11.0 seconds. From the policy performance graph shown below in figure 4.7, the 'Always Hold' policy was the worst performer and the 'Potential Fields' policy was the best performer with most episode durations occurring above 10.0 seconds.

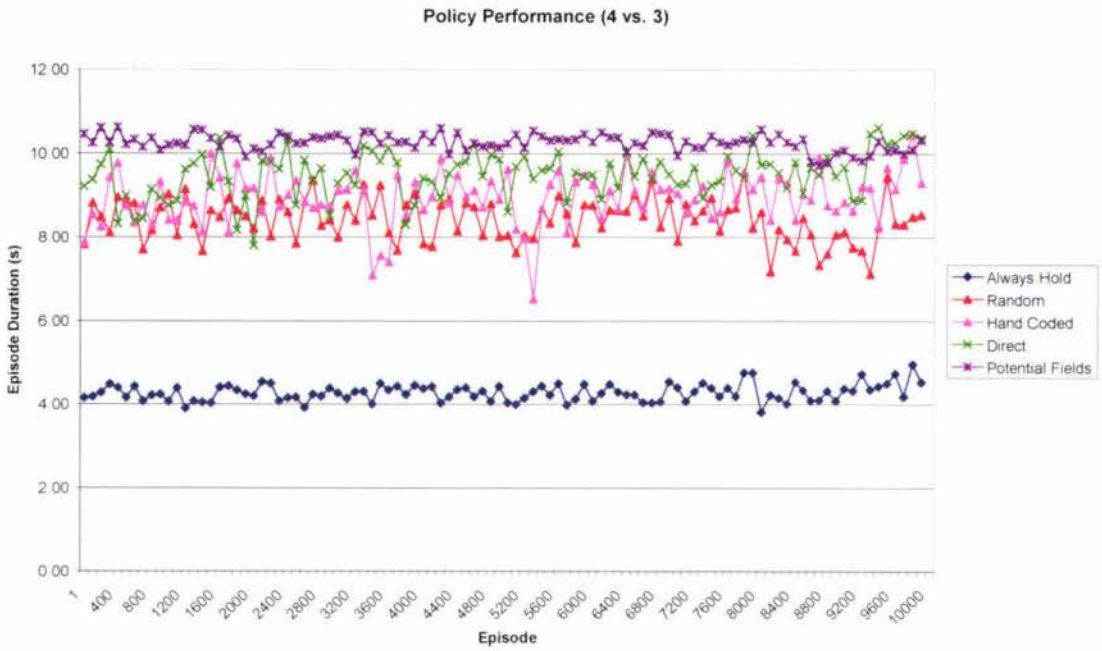



Figure 4.7: Policy performance graph for 4 vs. 3 keepaway.

As with the previous rounds of testing, the 'Random', 'Hand Coded' and 'Direct' policies have remained clustered, in this case between 8.0 and 10.0 seconds, however the performances appear to be much more ranged in their distributions than the previous rounds.

The box and whisker analysis presented in figure 4.8 below shows the 'Direct' policy's distribution has highest variation with the 'Always Hold' and 'Potential Fields' distributions remaining the smallest. These distributions continue to follow the pattern which emerged in the previous 3 vs. 2 and 2 vs. 1 result sets where all of the distributions are skewed to the left with the exception of the 'Potential Fields' distribution which is more symmetrical. The inter-quartile ranges also remain similar to the 3 vs. 2 results, although the inter-quartile ranges and medians are more spread out than the 2 vs. 1 result set.

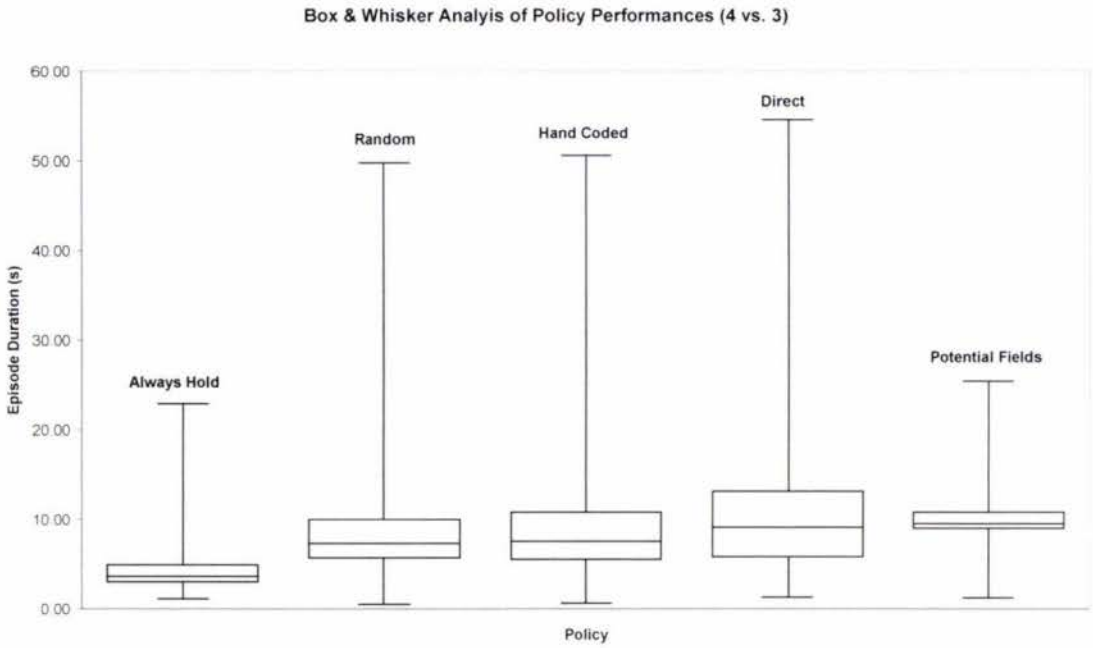



Figure 4.8: Box and whisker analysis of 4 vs. 3 keepaway soccer performance.

The episode outcomes of 4 vs. 3 keepaway saw the average 'taken' to 'out' ratio drop to approximately 1:840 with individual policy performances dropping to 325:675 for the 'Random' policy, 170:830 for the 'Hand Coded' policy, 140:860 for the 'Direct' policy, 100:900 for the 'Always Hold' policy and 65:935 for the 'Potential Fields' policy. These ratios are illustrated in the following graph.

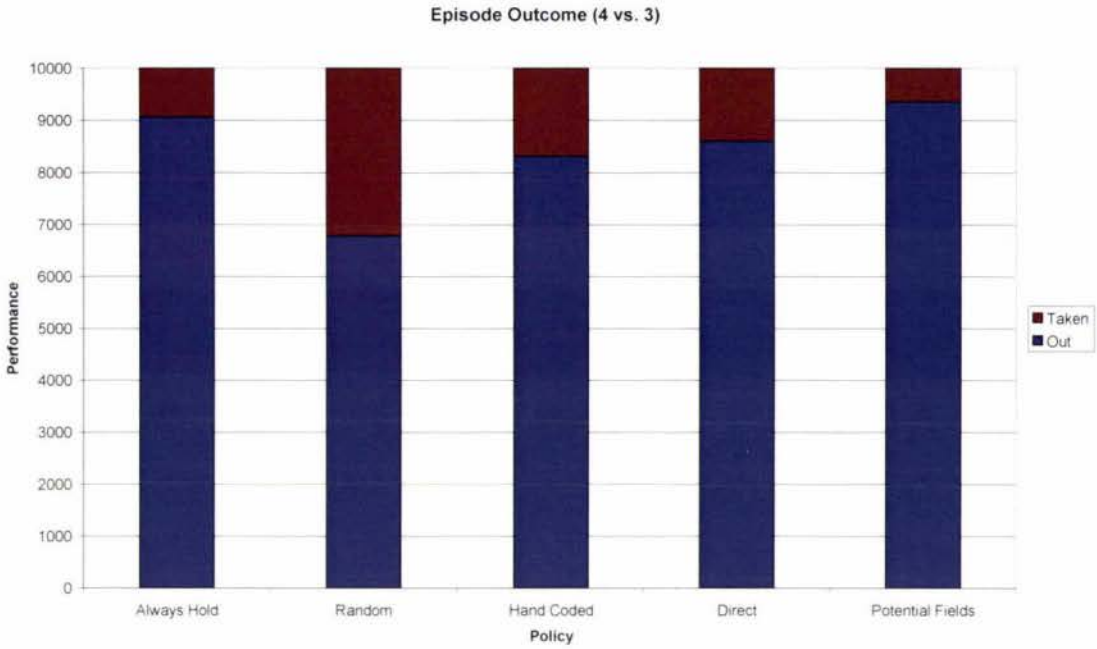



Figure 4.9: Outcome ratios for 4 vs. 3 policy performance.

The overall performance of the policies differed from the previous performance summaries as demonstrated in the following table.

| Policy | Mean | Standard Deviation |
|---|---|---|
| **Always Hold** | 4.3 | 1.9 |
| **Random** | 8.4 | 4.3 |
| **Hand Coded** | 8.9 | 5.1 |
| **Direct** | 9.5 | 1.9 |
| **Potential Fields** | 10.3 | 1.9 |

Table 4.6: Summary of results for 4 vs. 3 keepaway soccer.

This result set was the first to demonstrate the policy order which dominated the provided Framework results. The 'Hand Coded' policy finally out performed the 'Random' policy, which up until this point was the leading policy of those provided with the Keepaway Framework.

All policies performed better in 4 vs. 3 testing than they had in previous rounds, although some increases were only slight. The smallest standard deviation of 1.9 seconds was shared not only by the 'Potential Fields' and 'Always Hold' policies, also by the 'Direct' policy.

The 'Potential Fields' policy was once again the top performer with an average episode duration of 10.3 seconds, a 0.7 second increase in performance from 3 vs. 2, and the 'Always Hold' policy

remained the least effective with an average episode duration of only 4.3 seconds, also a 0.7 second increase from it's previous performance.

### 4.2.4 5 vs. 4 Keepaway

5 vs. 4 keepaway is played on a 30m by 30m playing field with one keeper at each top corner, the bottom right hand corner and the remaining keepers positioned in the middle of the field with the takers positioned in the bottom left hand corner.

The policy performances for this round of testing ranged from approximately 5.0 seconds to 12.0 seconds with most policy durations falling between 9.0 and 11.0 seconds.

From the policy performance graph shown below, the clustering between the three middle performing policies, 'Direct', 'Hand Coded' and 'Random' isn't as pronounced as previous rounds due to the 'Random' policy's performance dropping between episodes 4000 and 6000, the 'Hand Coded' policy's performance peaking over the same period and the 'Direct' policy maintaining a steady performance.

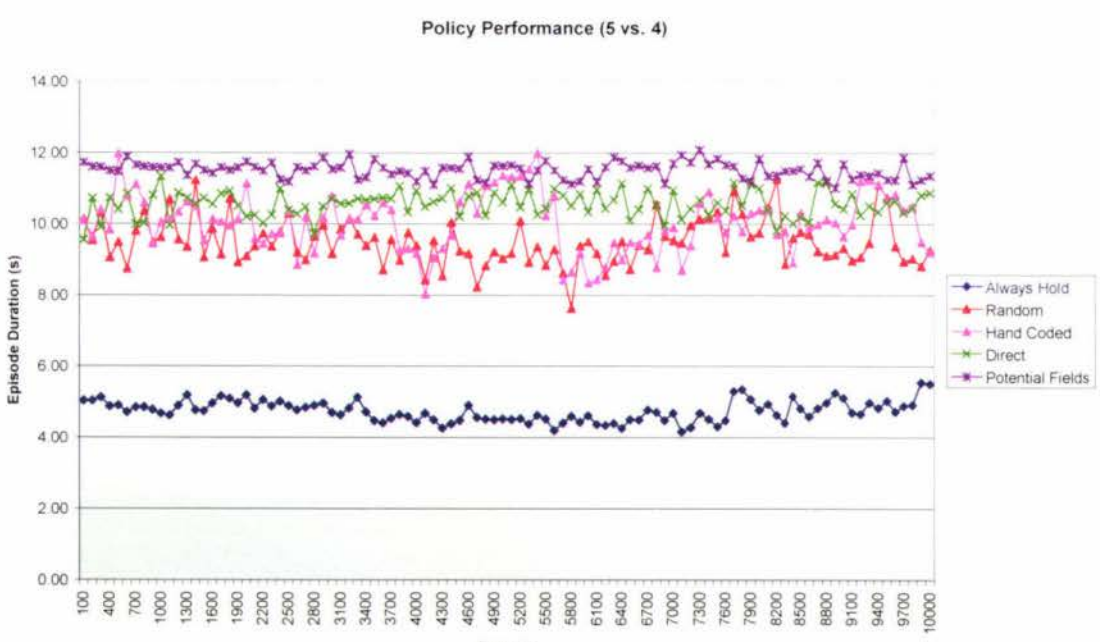



Figure 4.10: Policy performance graph for 5 vs. 4 keepaway.

The box and whisker analysis presented in the following figure shows the 'Hand Coded' policy to have the highest distribution and inter-quartile range, the 'Always Hold' policy to have the smallest

distribution and inter-quartile range and the 'Random' and 'Direct' policies sharing similar inter-quartile ranges.

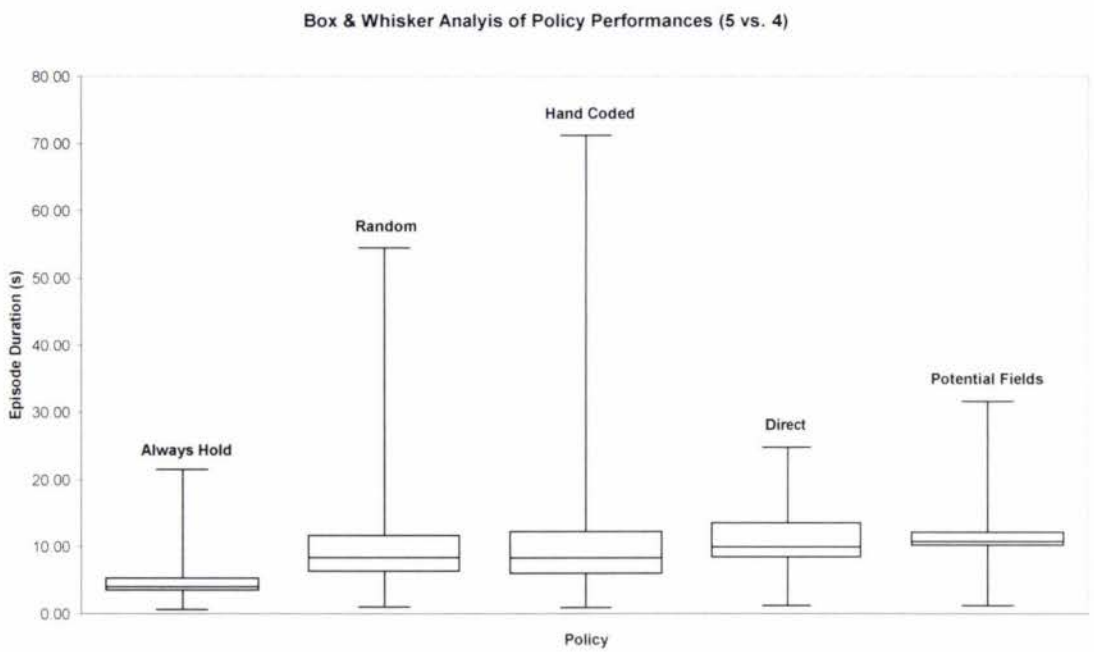



Figure 4.11: Box and whisker analysis of 5 vs. 4 keepaway soccer performance.

Consistent, for the most part, with the results collected from the previous 2 vs. 1, 3 vs. 2 and 4 vs. 3 test runs, the medians for all distributions are close together and exhibit an upward trend, most distributions remain skewed to the left with the exceptions of the 'Direct' and 'Potential Fields' distributions which are more or less symmetrical.

The episode outcomes for 5 vs. 4 exhibited below continues the general trend with more episodes ending in 'out' rather than 'taken' leading to an average ratio of approximately 220:780 'takens' to 'outs'. The results, illustrated in the graph below, shows the 'taken' to 'out' ratio drop to 420:580 for the 'Hand Coded' policy, 390:610 for the 'Random' policy, 130:870 for the 'Always Hold' and 'Potential Fields' policies, and 100:900 for the 'Direct' policy.

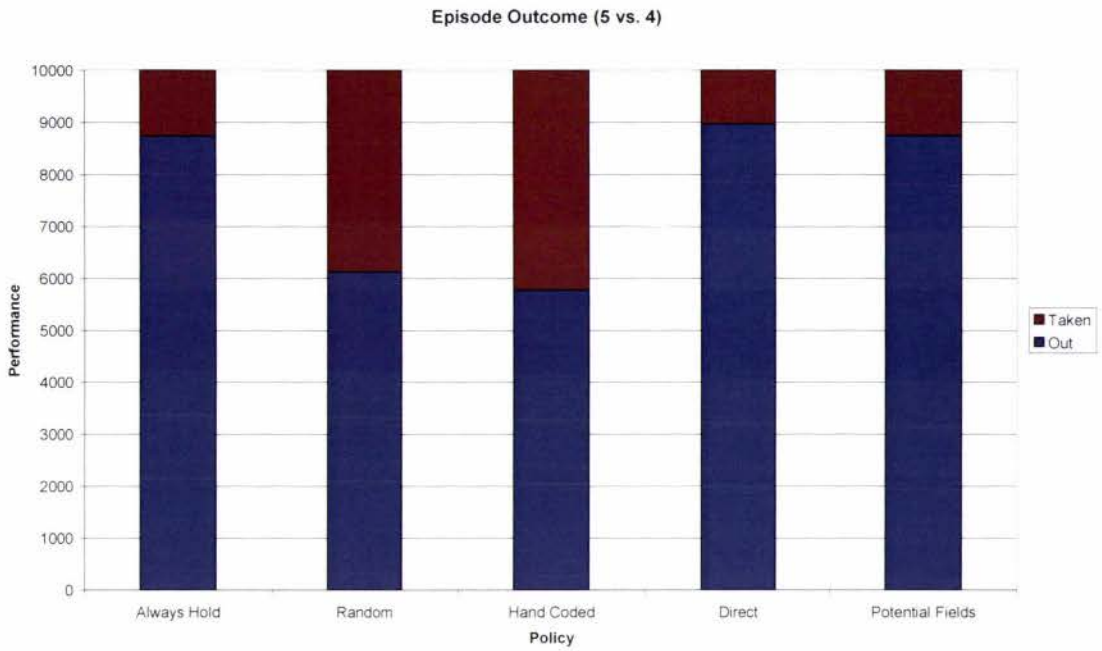



Figure 4.12: Outcome ratios for 5 vs. 4 policy performance.

The overall performance of the action-selection policies has been summarised in table 4.7 below.

| Policy | Mean | Standard Deviation |
|---|---|---|
| **Always Hold** | 4.7 | 2.0 |
| **Random** | 9.5 | 4.7 |
| **Hand Coded** | 10.0 | 5.9 |
| **Direct** | 10.5 | 3.1 |
| **Potential Fields** | 11.5 | 2.1 |

Table 4.7: Summary of results for 5 vs. 4 keepaway soccer.

As illustrated above, the 'Potential Fields' policy maintained best average performance but lost the smallest standard deviation of 2.0 seconds to the 'Always Hold' policy. The 'Random', 'Direct' and 'Hand Coded' policies performed similarly with means within half a second of each other, however, the 'Direct' policy had a significantly smaller standard deviation of 3.1 seconds.

The 'Potential Fields' policy once again proved itself to be the strongest performer, although the there was little between the three middle performers. The overall performance of the 'Direct' method saw it achieve results slightly ahead of the 'Hand Coded' policy and its standard deviation was considerably lower, meaning its results were more consistent.

For the three provided policies, 'Always Hold', 'Random' and 'Hand Coded', the 'Hand Coded' policy proved to be the best performer, followed by the 'Random' policy, then the 'Always Hold' policy.

This ordering is once again inline with the performance trend demonstrated in the provided Framework results.

Overall, for the last feasibility test run, both of the confidence model approaches have performed better than their provided counterparts and both with small standard deviations reinforcing the observation that these policies perform more consistently.

### 4.2.5 Summary

The overall results of the feasibility testing are summarised below in the following table.

| | 2 vs. 1 | | 3 vs. 2 | | 4 vs. 3 | | 5 vs. 4 | |
|---|---|---|---|---|---|---|---|---|
| **Policy** | **Mean** | **Std Dev** | **Mean** | **Std Dev** | **Mean** | **Std Dev** | **Mean** | **Std Dev** |
| **Always Hold** | 4.1 | 2.6 | 3.6 | 1.6 | 4.3 | 1.9 | 4.7 | 2.0 |
| **Random** | 4.0 | 2.6 | 8.2 | 4.3 | 8.4 | 4.3 | 9.5 | 4.7 |
| **Hand Coded** | 4.3 | 3.1 | 7.9 | 4.3 | 8.9 | 5.1 | 10.0 | 5.9 |
| **Direct** | 4.3 | 2.6 | 8.9 | 2.8 | 9.5 | 1.9 | 10.5 | 3.1 |
| **Potential Fields** | 4.6 | 1.6 | 9.6 | 1.6 | 10.3 | 1.9 | 11.5 | 2.1 |

Table 4.8: Policy performance summary for results collected using Helix.

The performance of each policy for each team configuration follows a linear pattern with most policies showing improvement on previous performances, the most notable exception to this rule is the 'Always Hold' policy's performance for 2 vs. 1 and 3 vs. 2 which dropped by 0.5 seconds between configurations. The graph below best illustrates these anomalies as well as the overall performance trends.

The other notable exception was observed during testing of 2 vs. 1 and 3 vs. 2 keepaway where the 'Always Hold' and 'Random' policies, respectively, out performed the 'Hand Coded' policy which was considered to be an unusual finding when compared to the results provided with the Framework.

Although these results were re-tested in additional runs of 10,000 episodes to confirm their validity, this trend persisted.

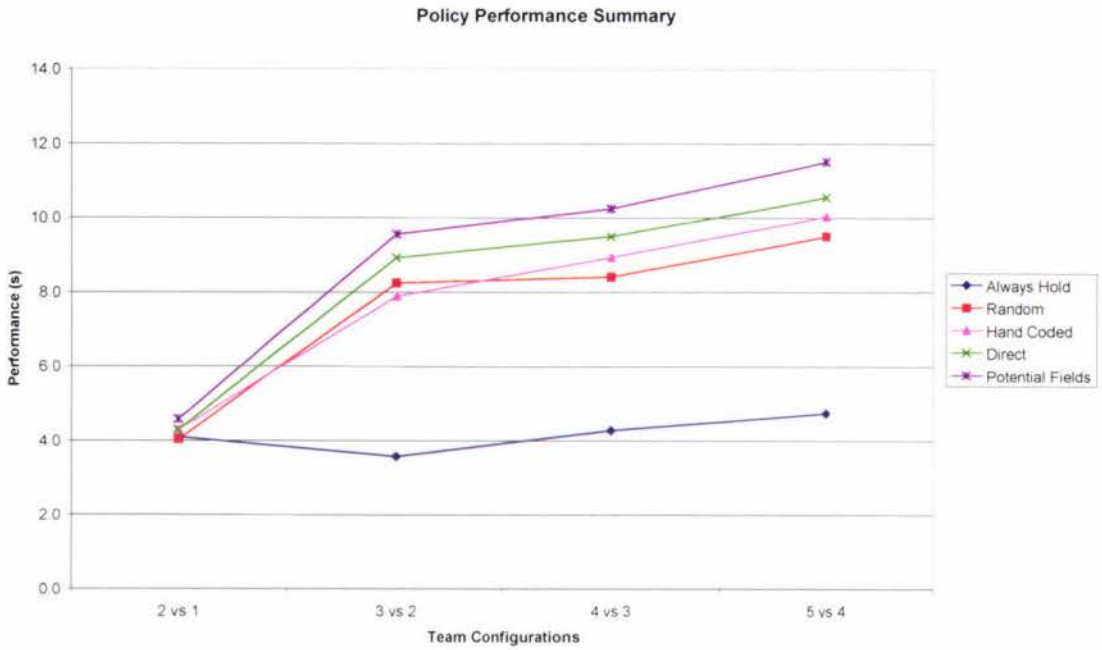



Figure 4.13: Policy performance summary for results collected using Helix.

As demonstrated, the 'Potential Fields' policy was the best overall performer, closely followed by the 'Direct' policy, with the 'Hand Coded', 'Random' and 'Always Hold' policies following the same performance pattern, on average, as was present in the provided Framework.

These test results prove that the confidence model approaches are indeed feasible action-selection policies when compared with the provided benchmark policies.

## 4.3 Scalability Testing

In order to test the scalability of the confidence model approach for the keepaway sub-domain of RoboCup simulated soccer, testing involved fielding specific team configurations on specific field sizes for 1,000 episodes per test.

Although 10,000 episode tests were found to be the optimum level to even out any performance anomalies, the time estimated to conduct the three test runs, 15 tests in total, is estimated to be in excess of the time required to conduct the entire feasibility and benchmark testing, making it impractical.

The results of this testing are detailed below with full page graphs available in section A2.2 of the Appendix.

### 4.3.1 3 vs. 2 Keepaway on Differing Field Sizes

The first tests to be conducted for scalability testing involve 3 vs. 2 keepaway played on differing field sizes. These tests were designed to examine how the performance of 3 vs. 2 keepaway is affected when players are in close quarters and are spread out over a larger area, both scenarios which are common in normal game situations. 3 vs. 2 was chosen as it can be plausibly scaled both down and up.

#### 4.3.1.1 15 Metre by 15 Metre Field

The first test involved 3 vs. 2 keepaway being played on a restricted field size of 15m by 15m, a decrease of 56%. As with standard 3 vs. 2 played on a 20m by 20m field, keepers are positioned in the top left and bottom left and right corners with the takers positioned in the bottom left corner. The keeper in the top left corner holds possession of the ball.

Episode durations ranged from 2.5 to 8.0 seconds, with the heaviest concentration of durations occurring between 5.0 and 7.5 seconds, as illustrated in the policy performance graph below. In order to graph all 1,000 results for each policy, each 50 episodes were averaged and those averages are shown in the performance graph below.

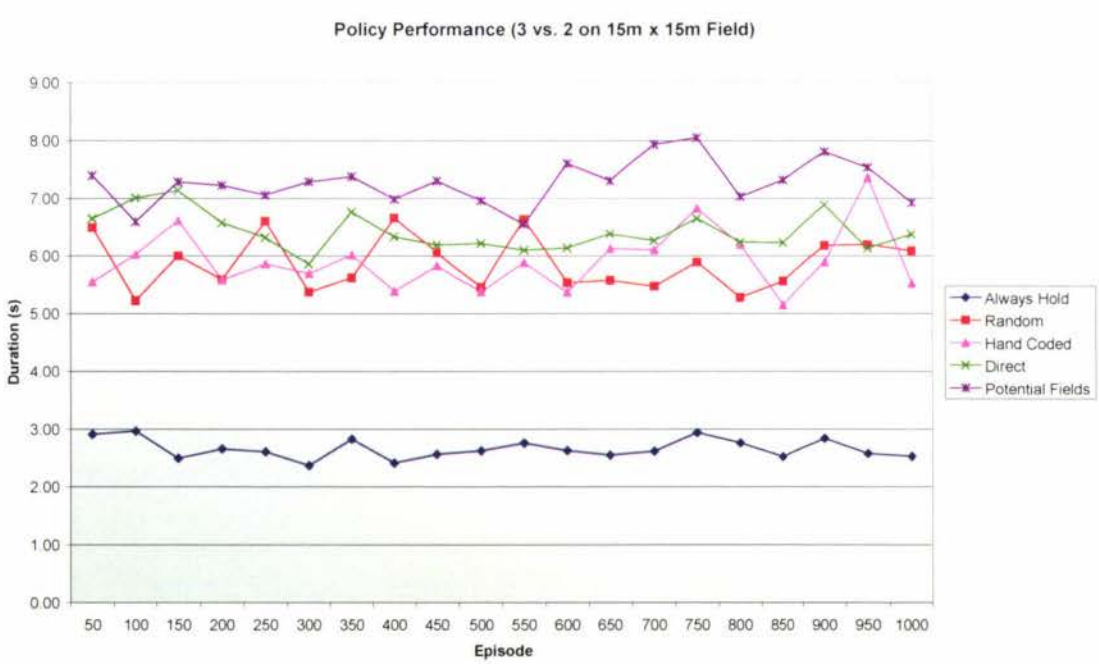



Figure 4.14: Policy performance graph for 3 vs.2 keepaway played on a 15m x 15m field.

As demonstrated, episode performances were chaotically ranged, with the 'Always Hold' and 'Potential Fields' policies clearly distinguishing themselves as the best and worst performers.

Further analysis shows that the inter-quartile ranges of the policies are fairly consistent. The medians for the distributions continued their upward trend, as observed in feasibility testing, and the trend towards left skewed distributions was also observed. These statistics have been graphed in the following box and whisker diagram.

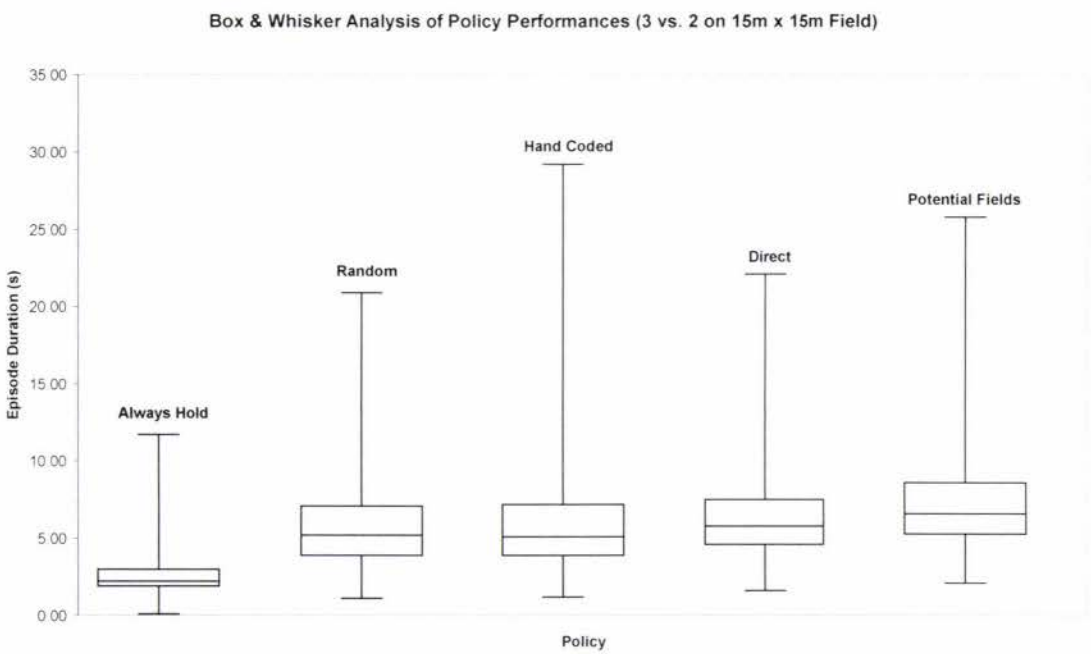



Figure 4.15: Box and whisker analysis of 3 vs.2 keepaway performance on a 15m x 15m field.

As shown, the 'Hand Coded' policy has the largest distribution and the 'Always Hold' policy, the smallest.

The episode outcomes for this test were also fairly close with an average ratio of 120:880 'takens' to 'outs' achieved. The 'Always Hold' policy performed with the least number of 'takens' at approximately 50:950 and the 'Direct' policy achieved the most at approximately 150:850. The remaining policies achieved ratios of 130:870 for the 'Random' policy, 125:875 for the 'Hand Coded' policy and 1:860 for the 'Potential Fields' policy. These are illustrated further in the following graph.

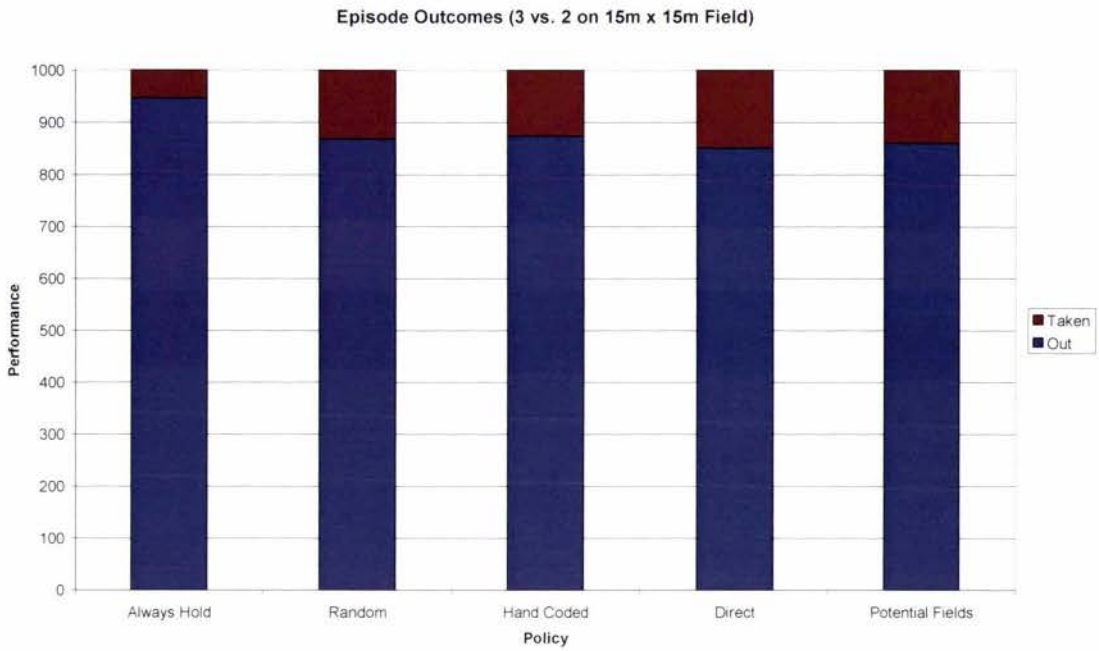



Figure 4.16: Outcome ratios for 3 vs. 2 policy performance on a 15m x 15m field.

The overall performance of the action-selection policies can be seen more clearly in table 4.9 below. The 'Always Hold' policy was the poorest performer averaging an episode duration of only 2.7 seconds, a further 0.9 second decrease in performance from it's lowest recorded average performance of 3.6 seconds for standard 3 vs. 2 keepaway. The 'Potential Fields' policy was the strongest performer averaging an episode duration of 7.3 seconds.

| Policy | Mean | Standard Deviation |
|---|---|---|
| **Always Hold** | 2.7 | 1.2 |
| **Random** | 5.9 | 2.9 |
| **Hand Coded** | 5.9 | 3.0 |
| **Direct** | 6.4 | 2.8 |
| **Potential Fields** | 7.3 | 3.0 |

Table 4.9: Summary of results for 3 vs. 2 keepaway soccer on a 15m x 15m field.

However, despite the trend of the 'Potential Fields' policy to exhibit the smallest deviation, for the restricted field size, the standard deviation was actually the highest at 3.0 seconds which was shared with the 'Hand Coded' policy. Also similar to the trend exhibited during 2 vs. 1 and 3 vs. 2 feasibility testing, the 'Random' policy and the 'Hand Coded' policy actually performed equally with a mean of 5.9 seconds.

As previously illustrated in the performance graph featured in figure 4.14 and outlined in the above table, the 'Potential Fields' policy was the best performer with the largest standard deviation and the

'Always Hold' policy was the poorest performer achieving both the lowest mean and standard deviation.

Overall, for the first test run, both of the confidence model approaches have performed better than their provided counterparts lending credence to the observation that these models handle play in confined spaces well.

#### 4.3.1.2 40 Metre by 40 Metre Field

The second test involved 3 vs. 2 keepaway being played on an enlarged field size of 40m by 40m, an increase of 400%. As with standard 3 vs. 2 played on a 20m by 20m field, keepers are positioned in the top left and bottom left and right corners with the takers positioned in the bottom left corner. The keeper in the top left corner holds possession of the ball.

Episode durations ranged from 6.0 to 30.0 seconds, with the heaviest concentration of durations between 15.0 and 25.0 seconds, as illustrated in the policy performance graph below.

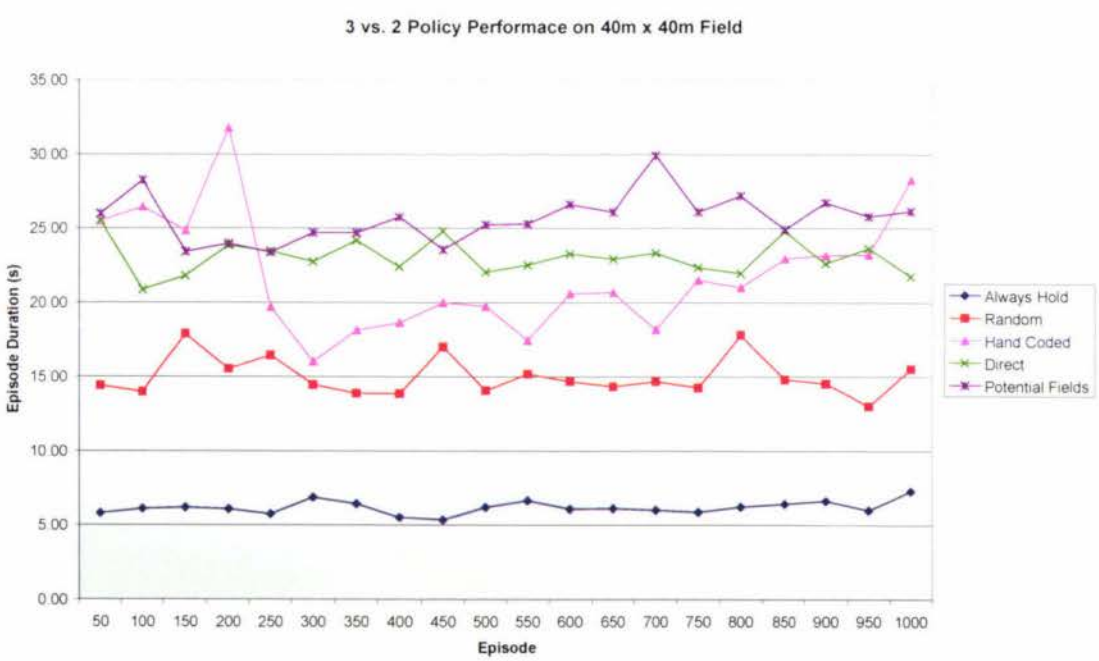



Figure 4.17: Policy performance graph for 3 vs.2 keepaway played on a 40m x 40m field.

As demonstrated, episode performances were widely spaced, with all policies clearly distinguishing themselves. Further analysis shows that the distributions of the policies are more widely ranged than previously observed. The medians of the distributions continued their upward trend, which is

noticeably more pronounced, as is the trend towards left skewed distributions. These statistics have been graphed in the following box and whisker diagram.

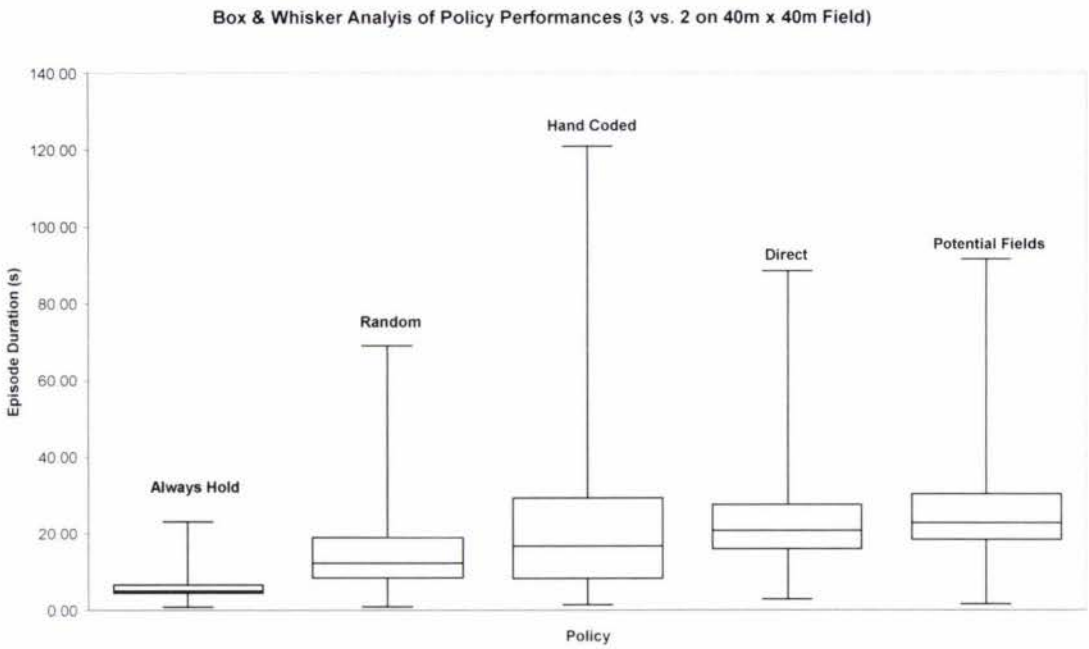



Figure 4.18: Box and whisker analysis of 3 vs.2 keepaway performance on a 15m x 15m field.

Consistent with the results observed for 3 vs. 2 on a 15 m x 15 m field, the 'Hand Coded' policy continues to display the largest distribution and the 'Always Hold' policy continues to display the smallest. The remaining distributions all display similar inter-quartile ranges and the 'Direct' and 'Potential Fields' policies also display similar deviations.

The episode outcomes for this test were also clearly distinguished with an average ratio of approximately 1:810 'takens' to 'outs' achieved. The 'Direct' policy performed with the least number of 'takens' at approximately 1:970, followed by the 'Always Hold' policy with a ratio of approximately 1:900, the 'Hand Coded' policy with 1:780, the 'Random' policy with 1:715 and finally the 'Potential Fields' policy achieved the lowest ratio of approximately 1:700. These are illustrated further in the following graph.

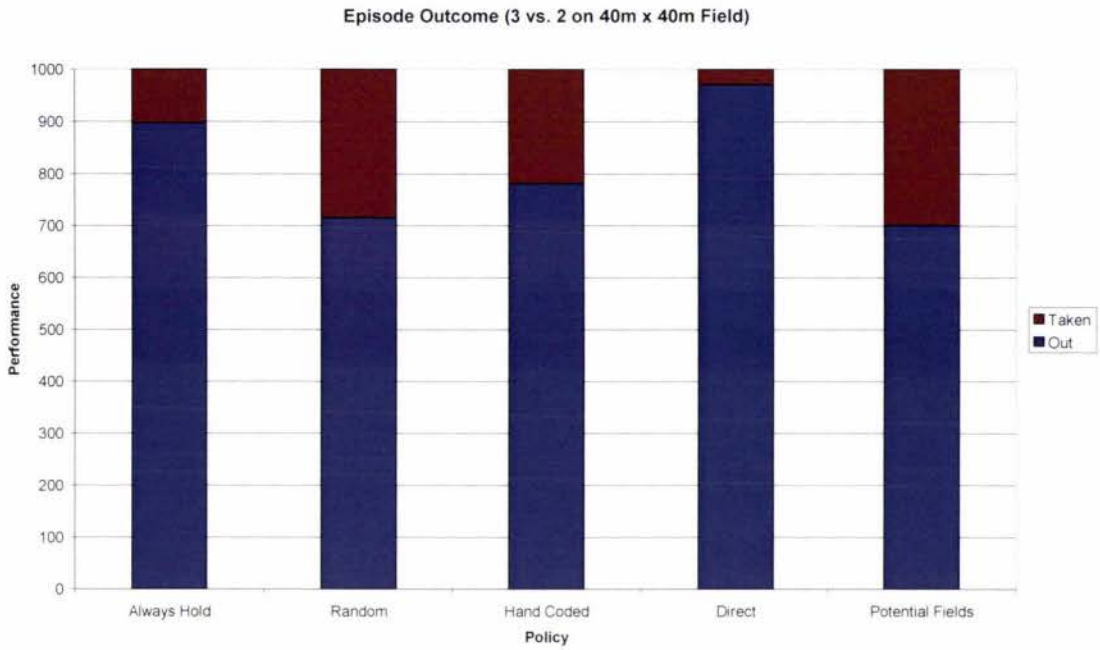



Figure 4.19: Outcome ratios for 3 vs. 2 policy performance on a 40m x 40m field.

The overall performance of the action-selection policies can be seen more clearly in table 4.10 below. The 'Always Hold' policy was the poorest performer averaging an episode duration of only 6.2 seconds, although its best performance so far, it was still well under the average policy performance of 18.4 seconds. The 'Potential Fields' policy was again the strongest performer averaging an episode duration of 25.7 seconds although the low standard deviation trend that this policy maintained during feasibility testing is once again compromised by the 11.1 seconds deviation recorded for this test, the second highest after the 'Hand Coded' policy which achieved a deviation of 18.1 seconds.

| Policy | Mean | Standard Deviation |
|---|---|---|
| **Always Hold** | 6.2 | 2.7 |
| **Random** | 15.0 | 10.0 |
| **Hand Coded** | 21.9 | 18.1 |
| **Direct** | 23.0 | 9.8 |
| **Potential Fields** | 25.7 | 11.1 |

Table 4.10: Summary of results for 3 vs. 2 keepaway soccer on a 40m x 40m field.

Over all these results follow the same trend exhibited during 4 vs. 3 and 5 vs. 4 feasibility testing, with the 'Potential Fields' and 'Direct' policies out performing the 'Hand Coded', 'Random' and 'Always Hold' policies the only anomaly being the lower deviation exhibited by the 'Random' policy.

Overall, for the second test run, both of the confidence model approaches have again performed better than their provided counterparts lending further credence to the observation that these models handle play over large unconfined spaces well.

### 4.3.2 11 vs. 11 Keepaway

The 11 vs. 11 keepaway test was designed to examine the performance of the confidence model policies by fielding 11 aside teams competing on the entire playing field in order to gauge how the policies would perform on a standard RoboCup Simulation team.

11 vs. 11 keepaway is played on a the entire 105m by 68m playing field with one keeper at each top corner, the bottom right hand corner and the remaining keepers positioned in the middle of the field with the takers positioned in the bottom left hand corner.

The policy performances for this round of testing ranged from approximately 11.0 seconds to 30.0 seconds with most policy durations falling between 20.0 and 27.0 seconds.

From the policy performance graph shown below, the clustering between the top four performing policies, 'Potential Fields', 'Direct', 'Hand Coded' and 'Random' is once again pronounced. While this clustering is similar to that observed in the results collected during the feasibility phase of testing, the clustering shown is less extensive.

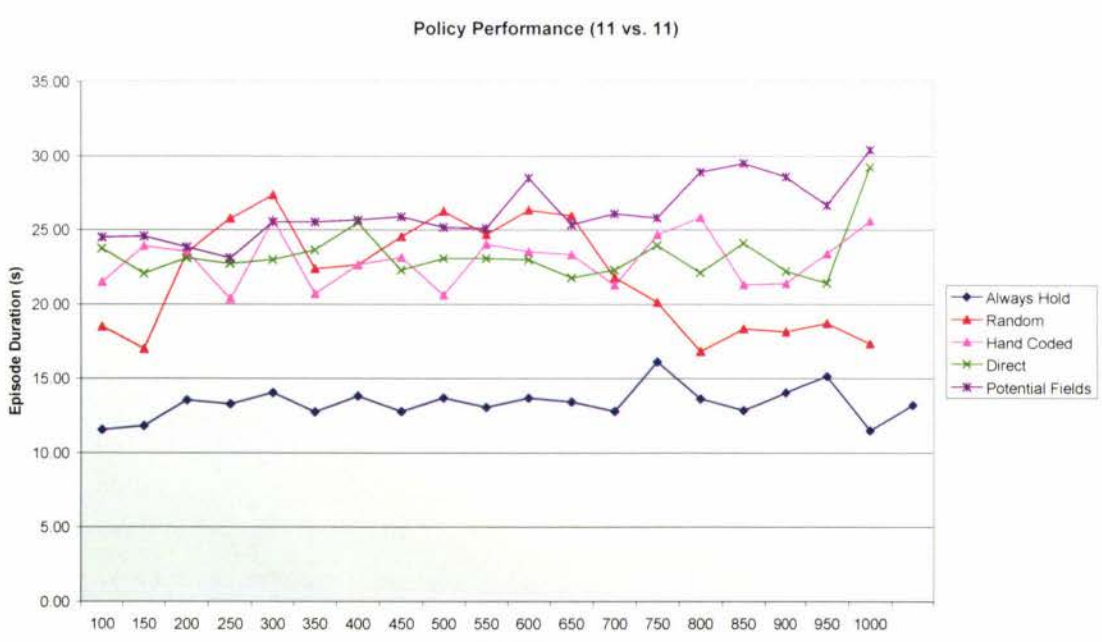



Figure 4.20: Policy performance graph for 11 vs.11 keepaway.

The box and whisker analysis presented in the following figure shows all of the policies exhibiting similar inter-quartile ranges, with the upwards median trend less pronounced than in previous results. The 'Direct' policy shows the largest distribution while the 'Potential Fields' policy shows the smallest. The 'Direct' policy, due to its large range is the most noticeably skewed and the 'Potential Fields' policy is nearly symmetrical. The remaining policies exhibit similar distributions and maintain the trend towards left skewing.

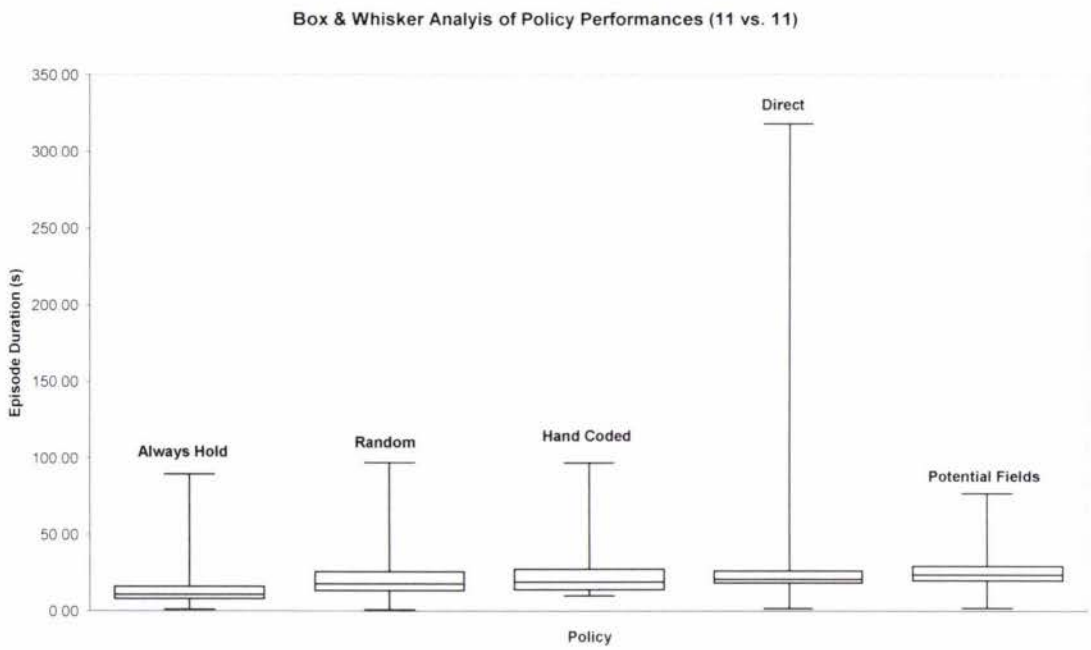



Figure 4.21: Box and whisker analysis of 11 vs.11 keepaway performance.

The episode outcomes for 11 vs. 11, as graphed below, buck the general trend observed during 2 vs.1, 3 vs.2, 4 vs. 3 and 5 vs. 4 feasibility testing where more episodes ended in 'out' rather than 'taken'. The average ratio of 'takens' to 'outs' dropped dramatically from 220:780 observed during 5 vs.4 testing to approximately 590:410 'takens' to 'outs'.

The results, illustrated below, show the 'taken' to 'out' ratios drop to 880:120 for the 'Hand Coded' policy, 875:125 for the 'Random' policy, 460:540 for the 'Potential Fields' policy, 395:605 for the 'Direct' policy and 348:652 for the 'Always Hold' policy.

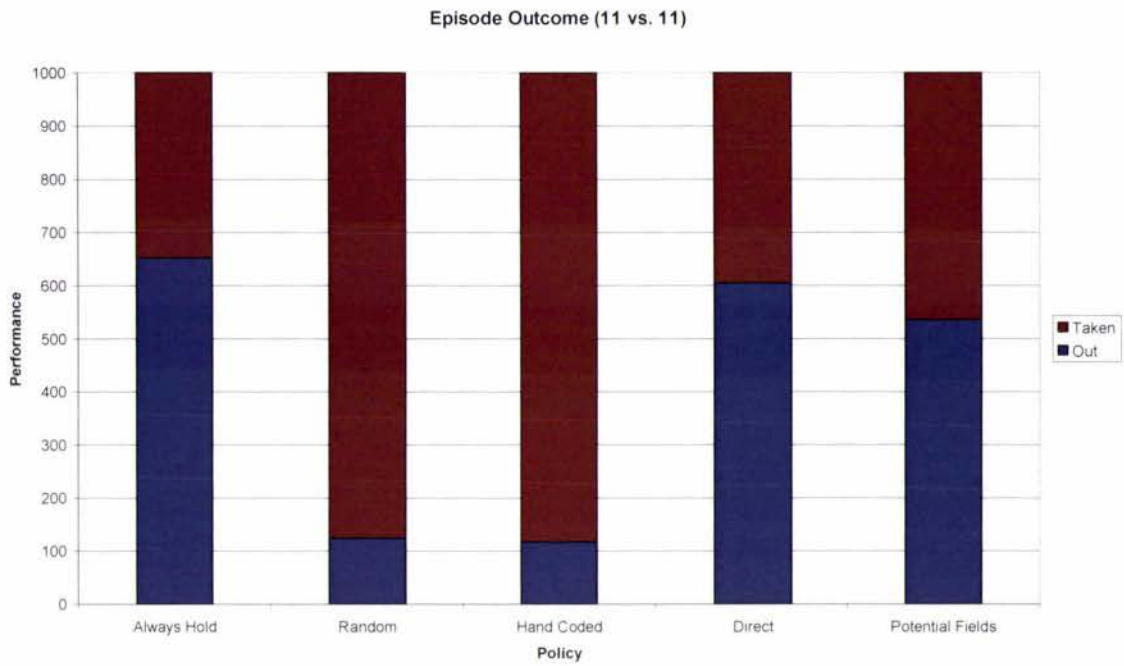



Figure 4.22: Outcome ratios for 11 vs. 11 policy performance.

The overall performance of the action-selection policies has been summarised in table 4.11 below.

| Policy | Mean | Standard Deviation |
|---|---|---|
| **Always Hold** | 13.3 | 7.5 |
| **Random** | 21.8 | 13.3 |
| **Hand Coded** | 22.9 | 12.8 |
| **Direct** | 23.3 | 11.5 |
| **Potential Fields** | 26.2 | 9.3 |

Table 4.11: Summary of results for 11 vs. 11 keepaway soccer.

As illustrated above, the 'Potential Fields' policy maintained the best average performance but lost the smallest standard deviation to the 'Always Hold' policy by 1.8 seconds. The 'Random', 'Hand Coded' and 'Direct' policies achieved a close average performance with their standard deviations showing an inverse pattern, to the previous trend, with the 'Random' policy achieving the highest standard deviation of 13.3 seconds.

The 'Potential Fields' policy once again proved itself to be the strongest performer, nearly 3.0 seconds clear of the 'Direct' policy which was the next best performer. For the three provided policies, 'Always Hold', 'Random' and 'Hand Coded', the 'Hand Coded' policy proved to be the best performer with an average of 22.9 seconds, followed by the 'Random' policy with 21.8 seconds then

the 'Always Hold' policy with 13.3 seconds. Once again these results are inline with the performance trend demonstrated in the provided Framework results.

Overall, for the last scalability test, both of the confidence model approaches have performed better than their provided counterparts and both with comparatively small standard deviations reinforcing the observation that these policies perform more consistently and capable of handling play on a full size field with a full complement of opponents and team mates.

### 4.3.3 Summary

Scalability testing saw three different aspects of the keepaway game tested, the first aspect involved observing how the action-selection policies handled play on a restricted field, the second aspect concerned observing how the policies performed in a larger region and the third, investigating how the policies performed using the full field with full team complements.

In order to evaluate the performances of these scalability policies, results were compared to those gathered from the benchmarking and feasibility testing.

The results for the 15m x 15m field size and 40m x 40m field size testing are compared below, in the following table, and performance graph with those collected from 3 vs. 2 feasibility testing.

| Policy | 20m x 20m Field | | 15m x 15m Field | | 40m x 40m Field | |
|---|---|---|---|---|---|---|
| | Mean | Std Dev | Mean | Std Dev | Mean | Std Dev |
| **Always Hold** | 3.6 | 1.7 | 2.7 | 1.2 | 6.2 | 2.7 |
| **Random** | 8.3 | 4.1 | 5.9 | 2.9 | 15.0 | 10.0 |
| **Hand Coded** | 7.7 | 3.9 | 5.9 | 3.0 | 21.9 | 18.1 |
| **Direct** | 8.2 | 3.4 | 6.4 | 2.8 | 23.0 | 9.8 |
| **Potential Fields** | 9.6 | 1.7 | 7.3 | 3.0 | 25.7 | 11.1 |

Table 4.12: Policy performance results collected during 3 vs. 2 feasibility testing and 3 vs.2 scalability testing.

As expected, policy performance for the 15m by 15m field showed a significant decrease from the policy performances achieved on the standard 20m by 20m field and policy performance for the 40m by 40m field was significantly increased.

The 'Random' and 'Always Hold' policies experienced a similar trend in performance with averages significantly down from the trend displayed by the remaining three policies, whose performances were very close.

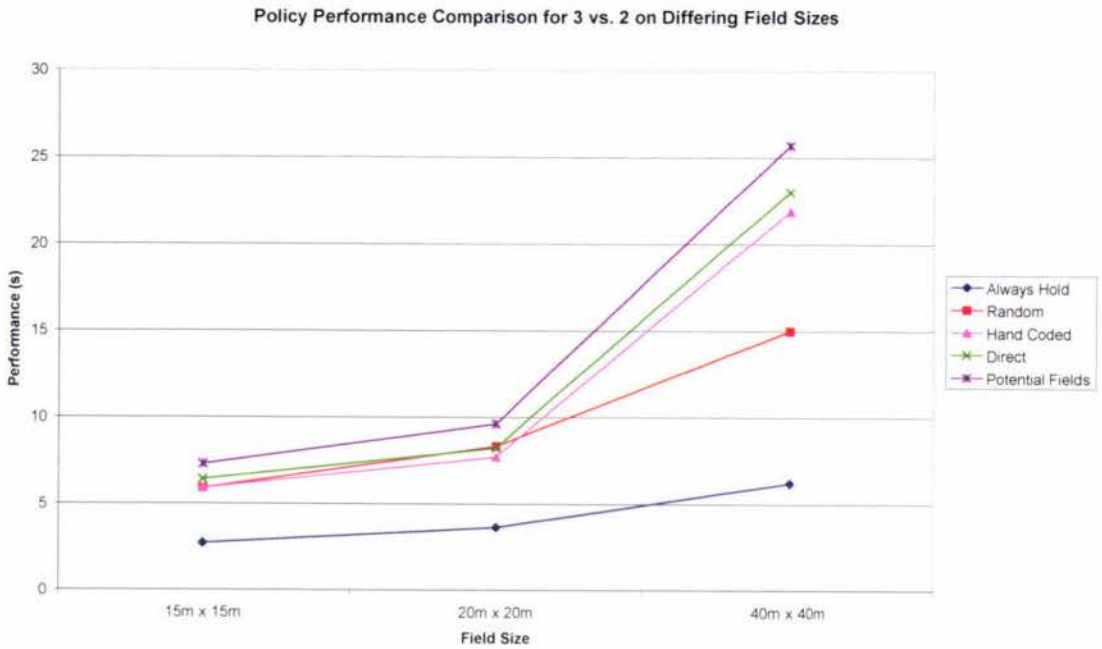



Figure 4.23: 3 vs. 2 policy performance comparison for differing field sizes.

These two trends demonstrate policies that display no regard for either their opponents positioning or their own team's positioning may still perform acceptably in small close quarter situations, such as many players clustering around the goal, but are distinctly left behind when play enters more open areas.

As previously stated, both the 'Potential Fields' and 'Direct' policies out performed the 'Always Hold', 'Random' and 'Hand Coded' policies provided by the Framework.

For 11 vs. 11 keepaway played over the entire field, results were also compared to those attained from the feasibility and benchmarking phases of testing.

As shown in the table and performance graph below, the performance for 11 vs. 11 saw a vast improvement over previous team configurations with even the 'Always Hold' policy achieving a reasonable average of 13.3 seconds.

| | 2 vs. 1 | | 3 vs. 2 | | 4 vs. 3 | | 5 vs. 4 | | 11 vs. 11 | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Policy** | **Mean** | **S.D.** | **Mean** | **S.D.** | **Mean** | **S.D.** | **Mean** | **S.D.** | **Mean** | **S.D.** |
| **Always Hold** | 4.1 | 2.6 | 3.6 | 1.6 | 4.3 | 1.9 | 4.7 | 2.0 | 13.3 | 7.5 |
| **Random** | 4.0 | 2.6 | 8.2 | 4.3 | 8.4 | 4.3 | 9.5 | 4.7 | 21.8 | 13.3 |
| **Hand Coded** | 4.3 | 3.1 | 7.9 | 4.3 | 8.9 | 5.1 | 10.0 | 5.9 | 22.9 | 12.8 |
| **Direct** | 4.3 | 2.6 | 8.9 | 2.8 | 9.5 | 1.9 | 10.5 | 3.1 | 23.3 | 11.5 |
| **Potential Fields** | 4.6 | 1.6 | 9.6 | 1.6 | 10.3 | 1.9 | 11.5 | 2.1 | 26.2 | 9.3 |

Table 4.13: Performance results collected during feasibility testing and 11 vs. 11 scalability testing.

The performance of each policy for each team configuration maintains the linear pattern observed during feasibility testing with all policies showing significant improvement on previous performances, unlike the results presented for the first and second phases of scalability testing, these results continued to exhibit the same trend with the 'Always Hold' policy clearly distinguishing itself from the remaining cluster of policies. The graph below best illustrates these clusters as well as the overall performance trends.

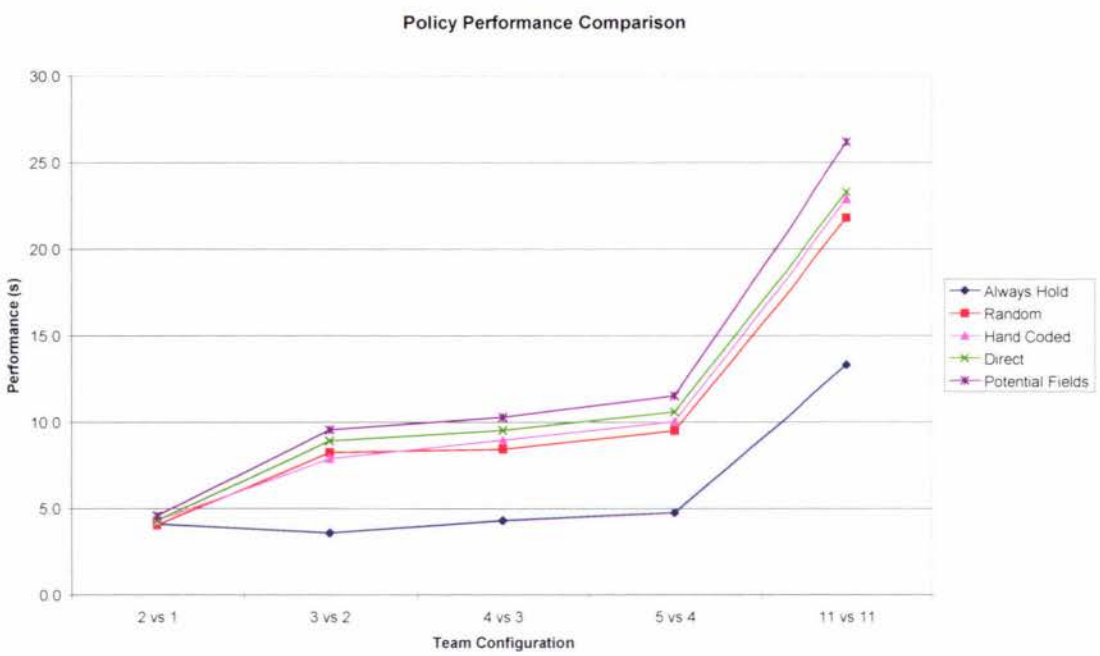



Figure 4.24: Policy performance comparison for differing team configurations.

As demonstrated, the 'Potential Fields' policy was the best overall performer for 11 vs. 11 keepaway, closely followed by the 'Direct' policy, with the 'Hand Coded', 'Random' and 'Always Hold' policies following the same performance pattern, as was present in the provided Framework.

These test results prove that the confidence model approaches are indeed scalable and able to manage play in both restricted quarters and over the entire field.

## 4.4 Entropy Testing

Entropy testing was included to help empirically evaluate the entropy of the confidence model approaches as, depending on which set of assumptions are used, the entropy equation chosen can give an extremely biased result.

The entropy tests conducted were aimed at empirically examining the internal performance of the keepers team running the 'Direct' policy to determine which set of assumptions should be applied to the entropy equations: the assumption that agents operating identical code are identical or; the assumption that agents operating identical code that are capable of performing different behaviours or the same behaviour differently, in the same time period given the same scenario are heterogeneous.

In order to conduct this testing, the confidence levels for the 'Passing' statistic were noted and averaged for each player at 100 episode intervals over a 1000 episode game. The statistics were initially set to 0 to ensure that no initial bias was introduced into the experiment and the 'Passing' statistic was chosen as it was observed that agents running the confidence model tended to favour this action.

The first test was conducted on a 'handicapped' 3 vs. 2 team running the 'Direct' policy aimed at fulfilling the requirement for the second set of entropy assumptions by replicating agents which could perform the same action differently although they run identical code.

In order to achieve this, a handicapping loop was included in the code which randomly chose a number from 0 to a given random seed number which was then used in a counting loop effectively pausing the agent until the count was complete. The results from this test are included below:

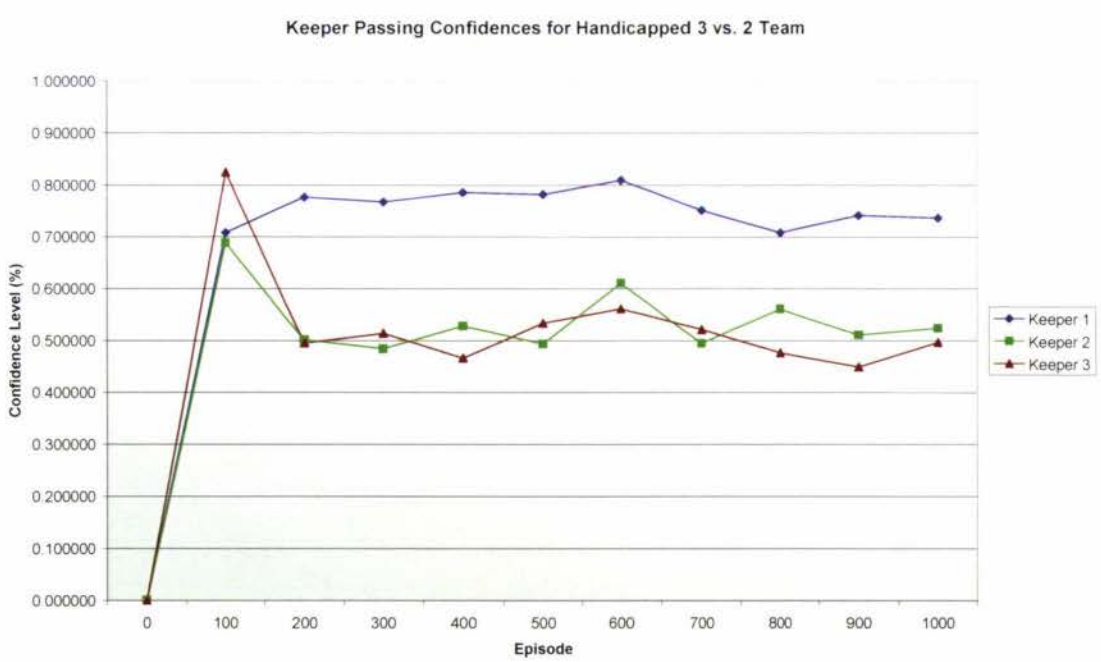



Figure 4.25: Keeper passing confidences observed for a handicapped 3 vs. 2 team.

As shown, the observed performances by each player are different, with one keeper averaging 76% passing confidence and the others 55% and 50% confidences respectively. These results were then compared to results collected from a normal, non-handicapped, game in order to determine, through a process of elimination, which set of assumptions matched the observed performance of the policies.

The next test was conducted using a standard 3 vs. 2 team playing takers running their default policy. As illustrated in the performance graph below, the competencies of the keepers were uniform with each keeper noting an approximate passing confidence of 86% for each of its peers.

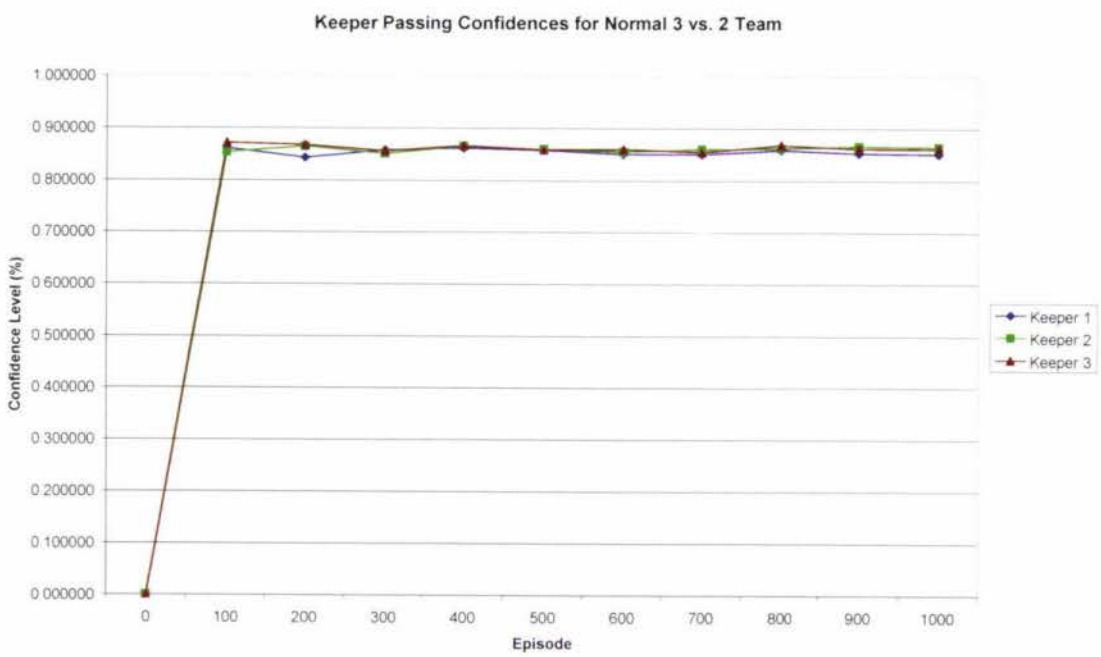



Figure 4.26: Keeper passing confidences observed for a 3 vs. 2 team.

By observing the results of this test against the handicapped test, it can be concluded that teams running the confidence model policies can be assessed for entropy using the first set of assumptions, that teams running identical code are identical, as this has been proven empirically through this comparison of performances.

# 5. Discussions & Conclusion

## 5.1 Discussion

This section documents the issues and resolutions observed during the course of this study.

### 5.1.1 Keepaway Soccer Framework

After a thorough investigation of research activities being conducted in the domains of robot soccer and robot rescue the Keepaway Framework, a sub-problem of simulated robot soccer, was chosen as the test bed for this study.

The primary attraction of the Keepaway Framework was the ability to allow direct comparisons between results as they are made up of a concrete performance indicator, the length of an episode. Other attractions included the ability to get up-to-speed quickly and the ability to compare results from your own policies with provided policies.

Although keepaway has been used for many years as a test bed for machine learning, 2005 was the first year which it was introduced as an easily implemented framework, and unfortunately this showed.

While it was fairly stable, relying on code exclusively written in C++ to handle the extensive threading required, often proved too much. This was especially evident in the all too frequent "Invalid Argument to ... " errors which, once traced, were determined to be the result of processes side-stepping numerous error checks and controls designed to inhibit this very behaviour.

For example, a frequent error received was "Invalid Argument to ActHandler" as a result of a method such as `Kick()` or `Dash()` being called with an out of bounds value, the equivalent of asking for a kick angle of 500 degrees or a dash speed of 1000 metres per second.

In order to reach the `ActHandler` method, which generated the error, the erroneous figure was first passed through an if-then statement in a high-level keepaway behaviour designed to limit the requested figure to the allowed maximum, then passed to another keepaway method designed to implement the required high-level behaviour in terms of a sequence of lower-level behaviours. The validity of the data is checked again at this stage using a similar if-then statement before passing the data to the Soccer Server which amalgamates all requested actions on an object i.e. if multiple players have requested to kick the ball, and generates a resultant vector and once again verifies

this data is correct before being passed to the `ActHandler` to action. The `ActHandler` then verifies the data again before actioning it, and in this case, returns the displayed error.

As no direct back-trace through the code could determine the reason why all of these numerous checks were somehow circumvented, the frequency with which this error occurred, and its tendency to hang the system, meant that while the Framework was running, the Soccer Server Monitor also had to be active to ensure that the simulation was still working.

The multi-threading code component was considered a logical source which could cause this type of randomly occurring problem and during development, many multi-threading pitfalls had to be worked around, including holding off the initialisation of the confidence model until the `StartEpisode()` code was called so that the required agent objects created by the Framework were given time to be initialised first.

These issues with multi-threading had the effect of causing many structures in the code to be repeated, causing redundancy, and complicated work arounds needing to be developed in order to ensure that the system had time to initialise the required objects or code before calls to them are made. The errors caused by the threads also severely slowed testing, as the system could not be trusted to run unsupervised, meaning testing was limited to 6 – 8 hours a day.

Although two newer versions, 0.5 and 0.6 of the Framework have been released while this study was completed, and some of these problems identified may have been fixed, before undertaking to use this Framework again, it would be useful to look at re-developing it using a language such as Java, which would be better suited for handling such a large and complex level of multi-threading.

Additionally, it would be useful to develop other simple policies which could be included and used to provide more meaningful comparisons and more realistic benchmarks. These could include policies based on hierarchical methods, reactive methods such as potential fields, learning methods such as Q-learning or reinforcement learning, and connectionist approaches such as neural networks.

### 5.1.2 Confidence Model Observations & Results

Takahashi noted that teams which used more collaborative play tactics such as 1-2 passing described in section 1.1 were inclined to perform better (Takahashi, 1999).

While the passing patterns found in keepaway did not require that level of complexity, a distinct passing pattern did assert itself. For want of a better description, this passing pattern resembles a

children's ball passing game called "piggy in the middle" where a ball is passed either clockwise or anticlockwise around a group of players and "the piggy in the middle" tries to intercept these passes. This pattern is best illustrated in the following diagram.

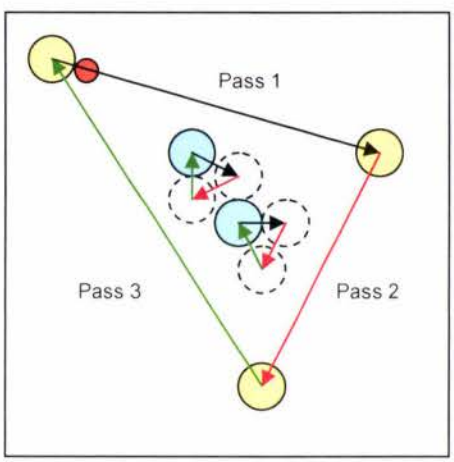



Figure 5.1: "Piggy in the middle" passing exhibited by players using the Potential Fields policy implementation.

While the ball is being passed around the group, the defenders are effectively pinned in the middle by their `GoToBall()` behaviours. This passing strategy, utilised fully by both confidence models, but used more effectively by the 'Potential Fields' policy, could be attributed to the strong performance trends exhibited by these policies.

During the development of the 'Direct' confidence model policy, it became apparent that the implementation suffered from the "local minima problem", previously identified as a short-fall of the Reinforcement Learning policy in section 3.1.4, where confidences became so high for one particular action that no other action was ever taken.

The solution to this problem is quite simple, by introducing a randomly chosen action at a frequency, controlled by a constant in the code, these minima are no longer formed.

Another spin off of introducing this 'choose random action' behaviour is that it is able to act as a 'random policy'. Not to be confused with the 'Random' policy provided in the Keepaway Framework, this type of random policy describes the process of exploration, by choosing random actions, which an agent using a learning policy like reinforcement learning may undertake, in order to be trained to know which actions lead to desirable outcomes and which lead to undesirable outcomes.

By increasing the frequency that random actions are chosen to once every step, this random policy was used to fast track the training period for the confidence model allowing training to be completed in less than 100 episodes as illustrated by the graphs in section 4.4. In both cases, confidence levels were initialised to 0 and by episode 100, the agents had developed a fair picture of their team-mates and opponents and were using the confidence levels to good effect.

In comparison, figures provided by Stone, Sutton & Kuhlmann showed that their tuned reinforcement learning policy required a training period of between 10,000 and 20,000 episodes depending on the complexity of the opponents faced (Stone et al, 2005a).

These results were released with the new version 0.6 of the Framework with updated benchmark figures as well. A major change with version 0.6, is the inclusion of a revised set of policies whose performance averages are compared in the table below. These figures were gathered by averaging the results of 3 vs. 2, 4 vs. 3 and 5 vs. 4 performances made on Helix and compared with the provided results posted on the Keepaway website (Stone et al, 2005b).

| | Version 0.4 | | Version 0.6 | |
|---|---|---|---|---|
| | **Mean** | **Std. Dev** | **Mean** | **Std. Dev** |
| **Always Hold** | 4.2 | 1.8 | 2.9 | 1.0 |
| **Random** | 8.3 | 4.4 | 5.3 | 1.8 |
| **Hand Coded** | 8.9 | 5.1 | 13.3 | 8.3 |

Table 5.1: Average performance comparison between version 0.4 and 0.6 of the Framework.

As shown, the performance of the tuned 'Hand Coded' policy has clearly improved, so much so that it out performs both the 'Direct' policy whose average performance is 9.6 seconds and the 'Potential Fields' policy whose average is 10.5 seconds. However, this 'Hand Coded' policy is in fact an implementation of reinforcement learning, so provides a good basis for comparing the confidence model to a more complex learning application.

Although the confidence model polices can be trained more rapidly and exhibit strong teamwork through passing collaboration, there can be no denying the strength of the various learning implementations for overall consistently high performance, although achieved at a much higher expense with relation to both time and effort involved in creation.

From a closer look at Stone, Sutton & Kuhlmann's results, an interesting trend was found amongst their scalability results. Presented in the following table are their results for 3 vs. 2 testing on a 15m x 15m field and a 20m x 20m field (their testing finished at 30m x 30m, so no comparison existed for the 40m x 40m scalability test).

| | Potential Fields | | Reinforcement Learning | |
|---|---|---|---|---|
| | **Mean** | **Std. Dev** | **Mean** | **Std. Dev** |
| **15m x 15m** | 7.3 | 3.0 | 7.4 | 0.9 |
| **20m x 20m** | 9.6 | 1.7 | 10.4 | 0.4 |

Table 5.2: Performance comparison between the 'Potential Fields' policy and 'Reinforcement Learning' policy for 3 vs. 2 keepaway over differing field sizes.

As illustrated, performance was reasonably consistent, which reinforces the findings of scalability testing in chapter 4, that the confidence model was indeed flexible and scalable in terms of team configurations and field sizes.

Although the standard deviations for the reinforcement learning algorithm are so low, it is assumed that this is due to some change in the internal makeup of the Framework after version 0.4 which was used for testing during this study and therefore these figures cannot be directly comparable.

## 5.2 Conclusion

To conclude this investigation, the research hypotheses posed in section 1.3 of this study have been re-examined and the findings summarised as follows:

1. *What is involved in the two large collaborative multi-agent domains of robot soccer and robot rescue? And which domain is better suited to conduct development and experimentation in?*

The domains of robot soccer and robot rescue were considered at length in chapter 2. Both robot soccer and robot rescue were found to be useful fields in which to investigate collaborative and emergent behaviour within synchronous and asynchronous competitive environments.

Robot soccer was considered to be the best domain in which to conduct this study within as there is a substantial interest in this domain, not only by academia, but also by the general public and as a suitable sub-problem existed, keepaway soccer, that could be used to get up to speed quickly and provided an excellent comparison framework.

2. *What is action-selection and what are the popular methods utilised in the chosen research domain?*

Action-selection was defined in chapter 3 as a method or methods used by an agent to determine the next course of action it wishes to perform within its environment.

Popular methods of action selection were also examined in this section and techniques such as Bayesian theory, where decisions are determined by balances of probabilities; neural networks where the decision process is the result of the constant adjustment of interactions between simple neuron like components; potential fields, a reactive approach utilising force fields; reinforcement learning, a popular method of machine learning where actions are reinforced through the receipt of rewards; and Q-learning, a refinement of reinforcement learning using Markov Decision Processes and discounted rewards, were all examined.

3. *Are considerations of team entropy noticeably included or excluded in the workings of these action-selection policies?*

From the study of the above action-selection methods, it was noted that the majority are action-centric and all exhibited a tendency to choose actions based solely on the merit of the action alone without taking into account the players involved in completing the action. In terms of entropy, it was clear from this observation that any considerations given for differing agent roles or strengths and weaknesses were excluded from consideration.

4. *How can an individual agent's strengths and weaknesses be assessed so that they can be compared and contrasted with its peers?*

Taking a lesson from human heuristic reasoning in section 3.2.2, it was determined that the best representation for the confidence model was as a percentage, represented as a double precision floating point number between 0.000 and 1.000, signifying how certain a player is in the abilities of it's peers and opponents.

Four statistics were then developed to represent this models application in the RoboCup soccer domain, these were Passing, Intercepting, Marking and Evading, each used to denote a players observed abilities in these four main skill areas. By combining these statistics together, as demonstrated in section 3.2.2.1, multifaceted equations can be formed which allow players to accurately gauge their peers and provide a medium for easy comparison.

During the course of a game, the observed confidences are saved periodically to a file, so at the end of the game, the individual performances of the agents can be averaged from their own self-assessment and the assessment of the players who interacted with them during the game. Because visual and aural information received during a game is noisy and limited and each agent may not collaborate with every other agent to the same degree, a complete picture of the game can be

formed in the same way, by combining each agents perceptions and calculating averages for the whole team.

5. *How can this method of assessing relative strength and weakness be implemented as an action-selection policy in the chosen domain?*

Section 3.2.2.2 detailed a direct approach whereby the confidence model equations were used to evaluate the players confidence in each proposed action, these confidences were then compared and the action with the highest confidence was then taken.

There were a number of initial issues with this implementation, including the issue of the "local minima problem" which was remedied by implementing a random action choice that was performed at a user defined frequency, and a "non-optimal passing problem" which was remedied through the introduction of potential fields theory into the policy.

This policy was tested in addition to the 'Direct' implementation and improved significantly on it's performance by using potential fields to control the level of influence a particular players confidence levels have in relation to those of opponents surrounding it. As stated, this saw an average episode duration increase of 0.9 seconds over all team sizes and field configurations.

6. *What is the actual entropy of a team using the above action-selection policies when it can be classed as both fully heterogeneous and fully homogeneous under both the bi-polar and social scales of entropy?*

The entropy of this approach was empirically tested in chapter 4.4 and findings showed that this approach to be homogeneous due to the convergence of the observed player confidence levels.

In order to conduct this experiment, a team using the 'Direct' policy were handicapped by a counting loop that made an agent pause for a random period of time during the instigation of a chosen action, therefore ensuring that players performing the action exhibited distinct performance competencies, as resultant passes were often misdirected or poorly timed. This handicapping was introduced to ensure that criteria required for social entropy were met.
The resulting confidence levels were then compared with levels recorded during a non-handicapped 3 vs. 2 game and the results proved conclusively that although the players changed roles and were able to choose different actions in the same time periods, their performances were still uniform.

*7. How does this method perform in comparison with other action-selection policies?*

Results of the empirical testing regime documented in chapter 3.3 were presented in chapter 4. These tests involved varying numbers of players being fielded on standard field sizes and included tests of 2 vs. 1, 3 vs. 2, 4 vs. 3 and 5 vs. 4 keepaway played on 15m x 15m, 20m x 20m, 25m x 25m and 30m x 30m fields respectively. Each of the five policies, the three provided Framework policies, 'Always Hold', 'Random' and 'Hand Coded' and the two confidence model policies, 'Direct' and 'Potential Fields', were tested for a period of 10,000 episodes and the outcomes were presented and discussed.

The results of testing proved that the performances of both the 'Direct' and 'Potential Fields' policies improved on the provided Framework policies by an average of 0.7 and 1.6 seconds respectively and that these policies maintained lower than average standard deviations as the level of attained performance was more reliable.

*8. How does this method scale in comparison with other action-selection policies?*

The third phase of the testing regime involved an investigation of how each confidence model policy performed over an expanded array of testing which included fielding a 3 vs. 2 team on differing field sizes and simulating a real game of 11 aside RoboCup simulated soccer by holding 11 vs. 11 keepaway over an entire field.

The results for these scalability tests were also positive and showed that both the 'Direct' and 'Potential Fields' policies performed well in both the cramped and spacious field sizes, once again averaging performances above the Framework polices by 0.5 and 1.4 seconds respectively for the 15m by 15m field size and 1.1 and 3.8 seconds respectively for the 40m by 40m.

The results for 11 vs. 11 keepaway also showed an improvement over the provided policies, this time by 0.4 and 3.3 seconds respectively.

These results proved that the confidence approach compared well with its peers and was indeed a scalable solution.

In conclusion, the confidence model was proposed as a player-centric action-selection methodology and was developed in two policies, the 'Direct' policy and the 'Potential Fields' policy which were implemented to play keepaway soccer.

Testing was conducted to determine the feasibility of this approach, as well as the scalability of the approach and these tests proved that these implementations were both viable and scalable action-selection methodologies for the domain of robot soccer keepaway.

The final phase of testing involved determining the entropy of this approach as there were many differing schools of thought as to what constituted a heterogeneous team and what constituted a homogeneous team. Through empirical testing, it was shown that this approach was homogeneous as all agents displayed the same amount of skill on the field during the course of a standard game.

Therefore, the confidence model implemented in the 'Direct' and 'Potential Fields' action-selection polices proved to be a sound, homogeneous solution, for a team wishing to implement a quickly trainable performance analysis solution.

## 5.3 Future Work

From the above discussion of this study, a number of recommendations for Future work were raised.

The first issue was the re-development of the Keepaway Framework using a language better suited to multi-threading, such as Java.

The second issue raised was the inclusion into the Framework of more advanced sample policies, such as a reinforcement learning policy or a neural network policy, which could then be used to further benchmark the confidence model.

Future work on the confidence model could therefore include further experimentation through hybridisation with other traditional action-selection policies until an optimum method is arrived at.

This optimum confidence model method could then be expanded to allow implementation on a RoboCup Simulation team and be tested in an 11 aside competition.

## 6. References


Blach, T. (2000a) Hierarchic Social Entropy: An Information Theoretic Measure of Robot Group Diversity, In *Autonomous Robots; Volume 8* (pp 209 – 237). The Netherlands: Kluwer Academic Press.

Balch, T. (2000b) *Teambots 2.0*. Retrieved April 8, 2005 from http://www.teambots.org/

Balch, T. & Parker, L. (Eds.) (2002) *Robot Teams: From Diversity to Polymorphism*. Canada: AK Peters.

BotSpot Website (n.d.) Retrieved September 10, 2005 from http://www.botspot.com/

Chen, M., Dorer, K., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., et al (2003) Users Manual RoboCup Soccer Server Version 7.07. Retrieved May 25, 2005 from http://sserver.sourceforge.net/docs/

Cyberbotics Webots 5 User Guide. (2004) Retrieved August 4, 2005 from http://cyberboticspc1.epfl.ch/cdrom/common/doc/webots/guide/guide.html

Di Pietro, A., While, L. & Barone, L. (2002) Learning in RoboCup Keepaway Using Evolutionary Algorithms. In W. Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, et al, (Eds.) *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference* (pp 1065-1072). New York: Morgan Kaufmann Publishers.

Dudek, G., Jenkin, M., Milios E., & Wilkes D. (1996) A Taxonomy for Multi-Agent Robotics. *Autonomous Robots*, 3(4), (pp 375 – 397). Canada: AK Peters.

Dudek, G., Jenkin, M., & Milios, E. (2002) A Taxonomy of Multirobot Systems. In T. Balch & L. Parker (Eds.), *Robot Teams: From Diversity to Polymorphism*. (pp 3 – 22).

FIRA Website (n.d.) Retrieved September 8, 2005 from http://www.fira.net/

Franklin, S. & Graesser, A. (1996) Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Lecture Notes in Computer Science; Volume 1193* (pp 21 – 35). London: Springer-Verlag.

Gerkey, B. & Mataric, M. (2004) On Role Allocation in RoboCup. In D. Polani, B. Browning, A Bonarini & K. Yoshida (Eds.) *RoboCup 2003: Robot Soccer World Cup VII* (pp 43 – 53). Berlin: Springer-Verlag.

Handwerk, B. (2003, March 11) Snakelike Robots May Fight Terror, Save Lives. *National Geographic News*. Retrieved November 5, 2005: National Geographic News: http://news.nationalgeographic.com/news/2003/01/0115_030115_snakebot.html

Hayes-Roth, B. (1995). An Architecture for Adaptive Intelligent Systems, *Artificial Intelligence*, 1-2(72), pp329 – 365.

Haugeland, J. (1985) *Artificial Intelligence: The Very Idea*. MA: MIT Press.

Jackson, P. (1999) *Introduction to Expert Systems*, (3rd Ed.). England: Addison-Wesley.

Kim, J. (2005) About FIRA. Retrieved September 8, 2005 from http://www.fira.net/about/greetings.html

Kitano, H. (Ed.) (1997) RoboCup-97: Robot Soccer World Cup I. Berlin: Springer-Verlag.

Kleiner, A., Brenner, M. & Brauer, T. (2005) Successful Search and Rescue in Simulated Disaster Areas. *RoboCup 2005: Robot Soccer World Cup IX*. (To appear)

Kuwata, Y. & Shinjoh, A. (2001) Design of RoboCup-Rescue Viewers – Towards a Real World Emergency System. In P. Stone, T. Balch & G. Kraetzschmar (Eds.) *RoboCup 2000: Robot Soccer World Cup IV* (pp 159 – 168). Berlin: Springer-Verlag.

Luger, G. (2002) *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. (4th Ed). USA: Addison Wesley.

Maes, P. (1994) Modeling Adaptive Autonomous Agents. *Artificial Intelligence*. 1(9), (pp135 – 162).

Maes, P. (1995) Artificial Life Meets Entertainment: Life like Autonomous Agents, *Communications of the ACM*, 38(11), (pp 108-114).

MacKay, D. (2003) *Information Theory, Inference and Learning Algorithms.* Cambridge, UK: Cambridge University Press. Retrieved December 20, 2005: David MacKay: Information Theory, Inference and Learning Algorithms Homepage: http://www.inference.phy.cam.ac.uk/mackay/itila/

Mackworth, A. (1993) On Seeing Robots. In A. Basu & X. Li, (Eds) *Computer Vision: Systems, Theory, and Applications*, (pp 1-13). Singapore : World Scientific Press.

Messina, E. (2003) *Measuring the Performance of Search and Rescue Robots.* Retrieved September 19, 2005 from http://www-users.cs.umn.edu/~stergios/icra2004mrsr-slides/LR-slidesMessina.pdf

Murphy, R. (2000) *Introduction to AI Robotics.* Cambridge, MA: MIT Press.

Murphy, R., Casper, J., & Micire, M. (2001) Potential Tasks and Research Issues for Mobile Robots in RoboCup Rescue. In P. Stone, T. Balch & G. Kraetzschmar (Eds.) *RoboCup 2000: Robot Soccer World Cup IV* (pp 339 – 344). Berlin: Springer-Verlag.

Ohta, M., Takahashi, T. & Kitano, H. (2001) RoboCup-Rescue Simulartion: In Case of Fire Fighting Planning. In P. Stone, T. Balch & G. Kraetzschmar (Eds.) *RoboCup 2000: Robot Soccer World Cup IV* (pp 351 – 356). Berlin: Springer-Verlag.

Padgham, L., Thangarajah, J., Poutakidis, D. & Fernando, C. (2002) Team Description for RMIT-on-Fire: RoboCup Rescue Simulation Team 2001. In A. Birk, S. Coradeschi & S. Tadokoro (Eds.) *RoboCup 2001: Robot Soccer World Cup V* (pp 755 – 757). Berlin: Springer-Verlag.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA: Morgan Kaufman Publishers.

Prokopenko, M. & Wang, P. (2004) Evaluating Team Performance at the Edge of Chaos. In D. Polani, B. Browning, A Bonarini & K. Yoshida (Eds.) *RoboCup 2003: Robot Soccer World Cup VII* (pp 89 – 101). Berlin: Springer-Verlag.

Russell, S. & Norvig, P. (1995) *Artificial Intelligence: A Modern Approach.* Englewood Cliffs, NJ: Prentice Hall.

RoboCup Website (n.d) Retrieved September 9, 2005 from http://www.robocup.org/

Sinclair. J.M (Ed.) (2000) *Collins English Dictionary* (4th Ed.) London: Collins.

Stone, P. & McAllester, D. (2001) An Architecture for Action Selection in Robotic Soccer. In *Proceedings of the Fifth International Conference on Autonomous Agents*, (pp. 316–323), New York: ACM Press.

Stone, P., Kuhlmann, G., Taylor, M. and Liu, Y. (2005) Keepaway Soccer: From Machine Learning Testbed to Benchmark. *RoboCup 2005: Robot Soccer World Cup IX*. (To appear).

Stone, P., Sutton, R. & Kuhlmann, G. (2005a) Reinforcement Learning for RoboCup-Soccer Keepaway. *Adaptive Behaviour*, 13(3), 165 – 188. Retrieved October 10, 2005 from http://www.cs.utexas.edu/users/pstone/Papers/bib2html/b2hd-AB05.html

Stone, P., Sutton, R. & Kuhlmann, G. (2005b) Learning to Play Keepaway. Retrieved October 8, 2005 from http://www.cs.utexas.edu/~AustinVilla/sim/keepaway/

Stone P., & Veleso M. (1999) Task Decomposition, Dynamic Role Assignment and Low-Bandwidth Communication for Real-Time Strategic Teamwork. *Artificial Intelligence*. 110(2), pp 432 – 441.

Strens, M. (2000) A Bayesian Framework for Reinforcement Learning. *Proceedings of the Seventh International Conference on Machine Learning* (pp 943 – 950) San Francisco, CA: Morgan Kaufmann Publishers.

Sutton, R. & Barto, A. (1998) *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Tadokoro, S. (2003) RoboCup Rescue Robot League: A Short Summary. In G. Kaminka, P. Lima & R. Rojas (Eds.) *RoboCup 2002: Robot Soccer World Cup VI* (pp 483 – 484). Berlin: Springer-Verlag.

Takahashi, T. (1999) LogMonitor: From Player's Action Analysis to Collaboration Analysis and Advice on Formation. In M. Veloso, E. Pagello & H. Kitano (Eds.) *RoboCup 1999: Robot Soccer World Cup III* (pp 103 – 113). Berlin: Springer-Verlag.

Takahashi, T. (2003) RoboCup Rescue Simulation League. In G. Kaminka, P. Lima & R. Rojas (Eds.) *RoboCup 2002: Robot Soccer World Cup VI* (pp 477 – 481). Berlin: Springer-Verlag.

Takahashi, T. & Naruse, T. (1999) From Play Recognition to Good Plays Detection: Reviewing RoboCup-97 Teams from Logfile. In M. Asada & H. Kitano (Eds.) *RoboCup-98: Robot Soccer World Cup II* (pp 187 – 192). Berlin: Springer-Verlag.

Tanaka-Ishii, K., Frank, I., Noda, I. & Matsubara, H. (2000) A Statistical Perspective on the RoboCup Simulator League: Progress and Prospects. In M. Veloso, E. Pagello & H. Kitano (Eds.) *RoboCup 1999: Robot Soccer World Cup III* (pp 114 – 127). Berlin: Springer-Verlag.

Trivedi, B. (2001, September 14) Search-and-Rescue Robots Tested at New York Disaster Site. *National Geographic News.* Retrieved November 5, 2005: National Geographic News: http://news.nationalgeographic.com/news/2001/09/0914_TVdisasterrobot.html

Walker T., Shavlik, J., & Maclin, R. (2004) Relational Reinforcement Learning via Sampling the Space of First-Order Conjunctive Features. In *Proceedings of the ICML Workshop on Relational Reinforcement Learning Banff, Canada* (pp 15 - 20). Retrieved December 27, 2005 from http://eecs.oregonstate.edu/research/rrl/papers/final/proceedings.pdf

Whiteson, S. & Stone, P. (2003) Concurrent Layered Learning. In *Second International Joint Conference on Autonomous Agents and Multiagent Systems*, (pp. 193 – 200), New York, NY: ACM Press.

Wooldridge, M. & Jennings, N. (1995). Agent Theories, Architectures, and Languages: A Survey. In Wooldridge and Jennings Eds., *Intelligent Agents* (pp 1 – 22). Berlin: Springer-Verlag.

# Appendix A1: User Guide and Code Printout

## A1.1 User Guide

### A1.1.1 Installation

Links to the software required and installation instructions are available from the Keepaway Framework Tutorial (Stone et al, 2005b).

For this implementation, the software installed was: the RoboCup Soccer Base and Soccer Server version 9.4.5; the RoboCup Soccer Server Monitor version 9.3.7 and; the Keepaway Framework version 0.4.

### A1.1.2 Configuration Settings

The simulation process is controlled by the `keepaway.sh` startup script which contains the configuration for the keepaway game. This configuration is divided into five categories, keeper options, taker options, client options, server options and trainer options. For these tests, only settings in the top four categories need to be configured including the number of keepers required, the number of takers required, field size, action-selection policies to be used and record keeping required. Additional fields within the LearningAgent code also need to be modified depending on the specific confidence model policy or the random action frequency required. These are marked in the code with double asterisks (**).

The following table demonstrates the settings required to run each of the five policies.

These are highlighted in red in the following excerpt of the sample startup script included in the next section.

| Setting | Always Hold | Random | Hand Coded | Direct | Potential Fields |
|---|---|---|---|---|---|
| keeper_learn | 0 | 0 | 0 | 1 | 1 |
| keeper_policy | "hold" | "rand" | "hand" | "learned" | "learned" |
| taker_learn | 0 | 0 | 0 | 0 | 0 |
| taker_policy | "hand" | "hand" | "hand" | "hand" | "hand" |
| launch_monitor | 1 | 1 | 1 | 1 | 1 |
| LearningAgent.cc | | | | cMode = DIRECT | cMode = POTENTIAL |

Table A1.1: Keepaway settings.

Other settings can be modified as required by the operator depending on the number of players, the size of field, the requirement for weights to be saved and loaded and the level of logging required. These are marked in blue in the following script.

Once the game configuration has been chosen and set, `LearningAgent.cc` will need to be recompiled using the `make` command from the keepaway `player` directory and then the startup script can then be run using the command `./keepaway.sh`.

### A1.1.3 Sample Startup Script

```
#!/bin/sh
# Keepaway startup script
# No commandline parameters.  All options are set in this file.
#
#source ~/.profile
# Top-level keepaway directory  **SET THIS OPTION**

keepaway_dir=/home/nsamara/keepaway-0.4

##############################################################
# Keeper options                                             #
##############################################################

num_keepers=3                    # number of keepers
keeper_load=0                    # should I load previously learned weights?
keeper_load_dir=                 # sub-directory of weight_dir where weights
                                   are stored
keeper_learn=1                   # should learning be turned on for keepers?
keeper_policy="learned"          # policy followed by keepers
#keeper_policy="hold"
#keeper_policy="hand"
#keeper_policy="rand"

##############################################################
# Taker options                                              #
##############################################################

num_takers=2                     # number of takers
taker_load=0                     # should I load previously learned weights?
taker_load_dir=                  # sub-directory of weight_dir where weights
                                   are stored
taker_learn=0                    # should learning be turned on for takers?
#taker_policy="learned"          # policy followed by takers
taker_policy="hand"

##############################################################
# Client options                                             #
##############################################################

save_weights=0                    # should I save learned weights
weight_dir=$keepaway_dir/weights  # top-level weight directory
save_client_log=0                 # should I save client logging info to a
                                    file?
log_level="1..1000"               # range of log levels to store
save_client_draw_log=0            # should I save client logged shape info to a
file?
```

```
client_log_dir=$keepaway_dir/logs # top-level client log directory
client_dir=$keepaway_dir/player   # directory containing player binary
client=keepaway_player            # name of player binary

#############################################################
# Server options                                            #
#############################################################

ka_width=20                       # Y-axis size of playing region
ka_length=$ka_width               # X-axis size of playing region

unrestricted_vision=0             # should I use 360-degree vision instead of
                                    90?
synch_mode=0                      # should I speed up with synchronous mode?

save_kwy_log=1                    # should I save episode info to .kwy file?
save_rcg_log=0                    # should I save game log to .rcg file?
save_rcl_log=0                    # should I save message lof to .rcl file?
log_dir=$keepaway_dir/logs        # directory to store kwy, rcg, and rcl logs

port=6000                         # server port used by players and monitor
coach_port=$[port + 1]            # server port used by offline trainer
olcoach_port=$[port + 2]          # server port used by online coach
sleep_time=8                      # time (in seconds) before starting PlayOn mode

launch_monitor=1                  # should I launch rcssmonitor on startup?

#############################################################################
# Trainer options  (Note: There is no trainer included with this release) #
#############################################################################

use_trainer=0                     # should I use a trainer instead of server
referee?
save_trainer_log=0                # should I save trainer's log a file?
trainer_dir=$keepaway_dir/        # directory containing trainer files
trainer=                          # name of trainer binary/class
```

## A1.2 Printout of LearningAgent.h Code

```
// ------------------------------------------------------------------------
// Header:    Learning Agent
// Author:    Samara Neilson
// Date:      2005
// Notes:     Based on code skeleton provided as part of Keepaway Framework by
//            Stone et al (2005b).
// ------------------------------------------------------------------------

#ifndef LEARNING_AGENT
#define LEARNING_AGENT

#include "SMDPAgent.h"
#include "BasicPlayer.h"
#include "KeepawayPlayer.h"
#include "Geometry.h"
#include "Parse.h"
#include "SoccerTypes.h"
#include "WorldModel.h"

// Constants
#define PASS 0
```

```
#define INTERCEPT 1
#define MARK 2
#define EVADE 3
#define TEAM_KEEPERS 0
#define TEAM_TAKERS 1
#define DIRECT 0
#define POTENTIAL 1
#define RAND_SEED 123456789

// ** Change to adjust frequency that random action is chosen
#define RANDOM_ACTION_FREQUENCY 100

// cModel structure represents the confidence model for each player in the game
struct cModel {
	double confidence;
	int successes;
	int total;
};

// cPlayer structure used to store confidence models
struct cPlayer {
	int myID;
	cModel stats[4];
};

// Learning Agent Class
class LearningAgent:public SMDPAgent
{
	// Private variables
	bool m_learning, m_saving;
	char m_saveWeightsFile[128];
	int m_lastAction;
	double m_lastState[MAX_STATE_VARS];

	WorldModel *WM;
	int myID;
	int episodeCounter;
	int cMode;
	bool handicapping;

	// Confidence model arrays
	cPlayer keepers[11];
	cPlayer takers[11];
	cPlayer kOrdered[11];
	cPlayer tOrdered[11];

	// Confidence model variables
	double P_Max;
	double I_Max;
	double E_Max;
	double M_Max;

	// Private functions and procedures
	void loadWeights(char *filename);
	void saveWeights(char *filename);

	void initialiseModels();
	int selectAction(double state[]);
	void sort(int type);

	int direct();
	int potentialFields(double state[]);
```

```
        void update(double state[], int action, double reward);

public:
        // Constructor
        LearningAgent(int numFeatures, int numActions, bool learning, char
*loadWeightsFile, char *saveWeightsFile, WorldModel *wm);

        // Public functions and procedures
        int startEpisode(double state[]);
        int step(double reward, double state[]);
        void endEpisode(double reward);
};

#endif
```

## A1.3 Printout of LearningAgent.cc Code

```
// ------------------------------------------------------------------------
// Class:       Learning Agent
// Author:      Samara Neilson
// Date:        2005
// Purpose:     Allows keepaway players to use the 'Direct' and 'Potential Fields'
//              action-selection policies which are part of the proposed
//              confidence model.
// Notes:       Based on code skeleton provided as part of Keepaway Framework by
//              Stone et al. (2005b).
// ------------------------------------------------------------------------

#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>

#include "LearningAgent.h"

// ------------------------------------------------------------------------
// Procedure:           Constructor
// Purpose:             Creates new instance of LearningAgent.
// Params In:           numFeatures   - Number of features in the keepaway state.
//                      numActions    - Number of available actions.
//                      learning      - Boolean representing if learning is active
//                                      or if player just uses already learned
//                                      weights.
//                      loadWeightsFile - File to load weights from.
//                      saveWeightsFile - File to save weights to.
//                                  wm - WorldModel, representing keepaway game.
// Params Out:          None.
// ------------------------------------------------------------------------
LearningAgent::LearningAgent(int numFeatures, int numActions, bool learning, char
*loadWeightsFile, char *saveWeightsFile, WorldModel *wm) : SMDPAgent(numFeatures,
numActions) {

        // Create an instance of LearningAgent
        m_learning = learning;
        strcpy(m_saveWeightsFile, saveWeightsFile);
        m_saving = (m_learning && strlen(saveWeightsFile) > 0);

        // Load weights file?
        if (strlen(loadWeightsFile) > 0) {
                loadWeights(loadWeightsFile);
```

```
        }

        // Initalise variables
        WM = wm;
        episodeCounter = 0;
        myID = 0;
        handicapping = false;

        // ** Choose action-selection policy to use
        cMode = DIRECT;
        // cMode = POTENTIAL;
}

// ------------------------------------------------------------------------
// Function:   Start Episode
// Procedure:  Start a keepaway episode.
// Params In:  state - array of doubles representing current state.
// Params Out: Integer representing action chosen.
// ------------------------------------------------------------------------
int LearningAgent::startEpisode(double state[])
{
        ObjectT me = WM->getAgentObjectType();
        myID = SoccerTypes::getIndex(me);

        // Finish initalisation
        if(episodeCounter == 0) {

                ObjectT me = WM->getAgentObjectType();
                myID = SoccerTypes::getIndex(me);
                initialiseModels();
        }

        // Choose first action
        m_lastAction = selectAction(state);
        memcpy(m_lastState, state, getNumFeatures() * sizeof(double));

        return m_lastAction;
}

// ------------------------------------------------------------------------
// Function:    Step
// Purpose:     Simulate the next step in the keepaway game.
// Params In:   state - array of doubles representing current game state.
// Params Out:  Integer representing action selected.
// ------------------------------------------------------------------------
int LearningAgent::step(double reward, double state[])
{
        // Last action ended well (as episode continues), update Confidence Model
        update(m_lastState, m_lastAction, 1.00);

        // Select next action
        m_lastAction = selectAction(state);
        memcpy(m_lastState, state, getNumFeatures() * sizeof(double));

        // Used for testing: agent pauses for random time before resuming
        if(handicapping) {
                int k = 0;
                int rNum = rand() % RAND_SEED;

                for(int i=0; i<rNum; i++) {
                        k = i;
                }
        }
```

```
        // Occasionally choose random episode to avoid local minima problem
        if((episodeCounter % RANDOM_ACTION_FREQUENCY) == 0) {
                m_lastAction = rand() % getNumActions();
        }

        return m_lastAction;
}

// ----------------------------------------------------------------------------
// Procedure:   End Episode
// Purpose:     Update the Confidence Models to reflect episode outcome.
// Params In:   None.
// Params Out:  None.
// ----------------------------------------------------------------------------
void LearningAgent::endEpisode(double reward)
{
        // Action resulted in episode end, update Confidence Model
        update(m_lastState, m_lastAction, -1.00);
        episodeCounter++;

        // Save weights every 10 cycles
        if((episodeCounter % 50 == 0)&&(episodeCounter < 1000)) {
                saveWeights(m_saveWeightsFile);
        }
}

// ----------------------------------------------------------------------------
// Procedure: Load Weights
// Purpose:           Loads existing Confidence Models.
// Params In: filename - name of file to load from.
// Params Out: None.
// ----------------------------------------------------------------------------
void LearningAgent::loadWeights(char *filename)
{
        fstream file_op(filename, ios::in);
        char * str;
        char line[256];
        double c = 0.0;
        int s = 0;
        int t = 0;

        // cout << myID << " : Load Models 1" << endl;

        // Load Keepers Confidence Models
        for(int i=0; i<WM->getNumKeepers(); i++) {
                for(int j=0; j<4; j++) {
                        file_op.getline(line, 256);
                        str = line;
                        c = Parse::parseFirstDouble(&str);
                        s = Parse::parseFirstInt(&str);
                        t = Parse::parseFirstInt(&str);

                        keepers[i].stats[j].confidence = c;
                        keepers[i].stats[j].successes = s;
                        keepers[i].stats[j].total = t;
                }
        }
        // cout << myID << " : Load Models 2" << endl;

        // Load Takers Confidence Models
        for(int i=0; i<WM->getNumTakers(); i++) {
                for(int j=0; j<4; j++) {
```

```
                    file_op.getline(line, 256);
                    str = line;
                    c = Parse::parseFirstDouble(&str);
                    s = Parse::parseFirstInt(&str);
                    t = Parse::parseFirstInt(&str);
                    takers[i].stats[j].confidence = c;
                    takers[i].stats[j].successes = s;
                    takers[i].stats[j].total = t;
                }
        }
        file_op.close();
}

// ------------------------------------------------------------------------
// Procedure: Save Weights
// Purpose:         Saves Confidence Models to a data file.
// Params In: filename - name of file to save data too.
// Params Out:      None.
// ------------------------------------------------------------------------
void LearningAgent::saveWeights(char *filename)
{
        char str[256];
        fstream output_file(filename, ios::out);

        // cout << myID << " : Save Models 1" << endl;

        // Save Keepers Confidence Models
        for(int i=0; i<WM->getNumKeepers(); i++) {
                sprintf(str, "%lf, %d, %d",
                        keepers[i].stats[PASS].confidence,
                        keepers[i].stats[PASS].successes,
                        keepers[i].stats[PASS].total);
                output_file << str << endl;

                sprintf(str, "%lf, %d, %d",
                        keepers[i].stats[INTERCEPT].confidence,
                        keepers[i].stats[INTERCEPT].successes,
                        keepers[i].stats[INTERCEPT].total);
                output_file << str << endl;

                sprintf(str, "%lf, %d, %d",
                        keepers[i].stats[MARK].confidence,
                        keepers[i].stats[MARK].successes,
                        keepers[i].stats[MARK].total);
                output_file << str << endl;

                sprintf(str, "%lf, %d, %d",
                        keepers[i].stats[EVADE].confidence,
                        keepers[i].stats[EVADE].successes,
                        keepers[i].stats[EVADE].total);
                output_file << str << endl;
        }

        // Save Takers Confidence Models
        for(int i=0; i<WM->getNumTakers(); i++) {
                sprintf(str, "%lf, %d, %d",
                        takers[i].stats[PASS].confidence,
                        takers[i].stats[PASS].successes,
                        takers[i].stats[PASS].total);
                output_file << str << endl;

                sprintf(str, "%lf, %d, %d",
```

```
                takers[i].stats[INTERCEPT].confidence,
                takers[i].stats[INTERCEPT].successes,
                takers[i].stats[INTERCEPT].total);
        output_file << str << endl;

        sprintf(str, "%lf, %d, %d",
                takers[i].stats[MARK].confidence,
                takers[i].stats[MARK].successes,
                takers[i].stats[MARK].total);
        output_file << str << endl;

        sprintf(str, "%lf, %d, %d",
                takers[i].stats[EVADE].confidence,
                takers[i].stats[EVADE].successes,
                takers[i].stats[EVADE].total);
        output_file << str << endl;
    }
    output_file.close();
}

// ------------------------------------------------------------------------
// Procedure:   Initalise Model
// Purpose:     Initalises the value of the Confidence Models.
// Params In:   None.
// Params Out: None.
// ------------------------------------------------------------------------
-
void LearningAgent::initialiseModels() {

    // Initialise Confidence Model for each keeper
    for(int i=0; i<WM->getNumKeepers(); i++) {
        keepers[i].myID = i;
        keepers[i].stats[PASS].confidence = 0.9;
        keepers[i].stats[PASS].successes = 0;
        keepers[i].stats[PASS].total = 0;

        keepers[i].stats[INTERCEPT].confidence = 0.5;
        keepers[i].stats[INTERCEPT].successes = 0;
        keepers[i].stats[INTERCEPT].total = 0;

        keepers[i].stats[MARK].confidence = 0.0;
        keepers[i].stats[MARK].successes = 0;
        keepers[i].stats[MARK].total = 0;

        keepers[i].stats[EVADE].confidence = 0.4;
        keepers[i].stats[EVADE].successes = 0;
        keepers[i].stats[EVADE].total = 0;
    }

    // Initalise a model for each taker
    for(int i=0; i<WM->getNumTakers(); i++) {
        takers[i].myID = i;
        takers[i].stats[PASS].confidence = 0.0;
        takers[i].stats[PASS].successes = 0;
        takers[i].stats[PASS].total = 0;

        takers[i].stats[INTERCEPT].confidence = 0.2;
        takers[i].stats[INTERCEPT].successes = 0;
        takers[i].stats[INTERCEPT].total = 0;

        takers[i].stats[MARK].confidence = 0.2;
        takers[i].stats[MARK].successes = 0;
        takers[i].stats[MARK].total = 0;
```

```
                takers[i].stats[EVADE].confidence = 0.0;
                takers[i].stats[EVADE].successes = 0;
                takers[i].stats[EVADE].total = 0;
        }

        // Initialise maximum values
        P_Max = 0.0;
        I_Max = 0.0;
        E_Max = 0.0;
        M_Max = 0.0;
}

// ------------------------------------------------------------------------
// Function: Select Action
// Purpose:  Allows player with the ball to determine the best action to take.
// Params In: state - array of doubles representing current state.
// Params Out: integer representing action selected, 0 represents hold, 1 pass
//             to keeper 1, 2 pass to keeper 2 etc.
// ------------------------------------------------------------------------
int LearningAgent::selectAction(double state[])
{
        int action = 0;

        // Sort Confidence Models to reflect closest-to sorting of Keepers and
        Takers

        sort(TEAM_KEEPERS);
        sort(TEAM_TAKERS);

        /* If I have the ball step unneccessary as this method is only called if
        keeper has the ball */

        // Decide if to pass ball to team-mate or hold onto it?
        switch(cMode) {
                case DIRECT: action = direct(); break;
                case POTENTIAL: action = potentialFields(state); break;
        }

        return action;
}

// ------------------------------------------------------------------------
// Procedure: Sort
// Purpose:   Sort Confidence Models to reflect order of state variables so
//            accurate statistics recording is possible.
// Params In: type - Integer representing the type (or side) of the team being
//                   sorted.
// Params Out:      None.
// ------------------------------------------------------------------------
void LearningAgent::sort(int type) {

        int index;
        ObjectT player = WM->getClosestInSetTo(OBJECT_SET_TEAMMATES, OBJECT_BALL);
        int numK = WM->getNumKeepers();
        int numT = WM->getNumTakers();

        // cout << myID << " ";  // " : Sort Models Start" << endl;

        // Sort Keeper objects to reflect Keepaway state order
        if(type == TEAM_KEEPERS) {

                // Sort list of keepers from the closest to the ball to the furthest
```

```
            ObjectT temp[numK];

            for (int i=0; i <numK; i++) {
                  temp[i] = SoccerTypes::getTeammateObjectFromIndex(i);
            }

            double WB_dist_to_K[numK];
            WM->sortClosestTo(temp, numK, player, WB_dist_to_K);

            // Refresh the data in the ordered array
            for(int i=0; i<numK; i++) {
                  index = SoccerTypes::getIndex(temp[i]);
                  kOrdered[i] = keepers[index];
            }

      // Sort Taker objects to reflect Keepaway state order
      } else {

            // Sort list of Takers from closest to the ball to furtherest
            ObjectT temp[numT];

             for (int i=0; i <numT; i++)
                  temp[i] = SoccerTypes::getOpponentObjectFromIndex(i);

            double WB_dist_to_T[numT];
            WM->sortClosestTo(temp, numT, player, WB_dist_to_T);

            // Refresh the data in the ordered array
            for(int i=0; i<numT; i++) {
                  index = SoccerTypes::getIndex(temp[i]);
                  tOrdered[i] = takers[index];
            }
      }
}

// -------------------------------------------------------------------------
// Procedure: Direct
// Purpose:   Determines confidence levels for all actions and chooses action
//            with highest confidence.
// Params In: None.
// Params Out: Integer representing chosen action.
// -------------------------------------------------------------------------
int LearningAgent::direct() {

      P_Max = 0.0;
      double max = 0.0;
      int action = 0;
      int numK = WM->getNumKeepers();

      for(int i=1; i<numK; i++) {

            // Determine which passing action has the largest confidence value
            max = kOrdered[0].stats[PASS].confidence +
                  kOrdered[i].stats[INTERCEPT].confidence;
            max = max - tOrdered[0].stats[INTERCEPT].confidence;

            if(max >= P_Max) {
                  P_Max = max;
                  action = i;
            }

            // Is the highest passing action better than the hold action?? If
```

```
            so, hold the ball
            if(P_Max <= kOrdered[0].stats[EVADE].confidence) {
                  action = 0;
            }
      }

      return action;
}

// ------------------------------------------------------------------------
// Procedure: Potential Fields
// Purpose:   Uses potential fields to determine which action has the largest
//            confidence level.
// Params In: state - array of doubles representing current game state.
// Params Out: Integer representing chosen action.
// ------------------------------------------------------------------------
int LearningAgent::potentialFields(double state[]) {

      // Initialise potential fields
            int action = 0;
            int numK = WM->getNumKeepers();
            int numT = WM->getNumTakers();
            Circle keeperFields[numK];
            Circle takerFields[numT];
            double radius;
            VecPosition centre;
            double overlap;
            double area;
            ObjectT temp;


            // Get positions of all keepers and takers and create fields for
            them
            for(int j=0; j<numK; j++) {

                  int index = (numK + numT) -1;
                  temp = SoccerTypes::getTeammateObjectFromIndex(j);

                  // Circle is centered around the current position of the
                  keeper
                  centre = WM->getGlobalPosition(temp);

                  // Radius of keeper field is the distance from keeper to
                  team-mate * confidence level
                  if(j == 0) {
                        radius = kOrdered[j].stats[EVADE].confidence;
                  } else {
                        radius = state[index+j] *
(kOrdered[0].stats[PASS].confidence + kOrdered[j].stats[INTERCEPT].confidence);
                  }

                  keeperFields[j].setCircle(centre, radius);
            }

            for(int j=0; j<numT; j++) {

                  int index = ((numK * 2) + numT) -1;
                  temp = SoccerTypes::getOpponentObjectFromIndex(j);

                  // Circle is centered around the current position of the
                  taker
                  centre = WM->getGlobalPosition(temp);
```

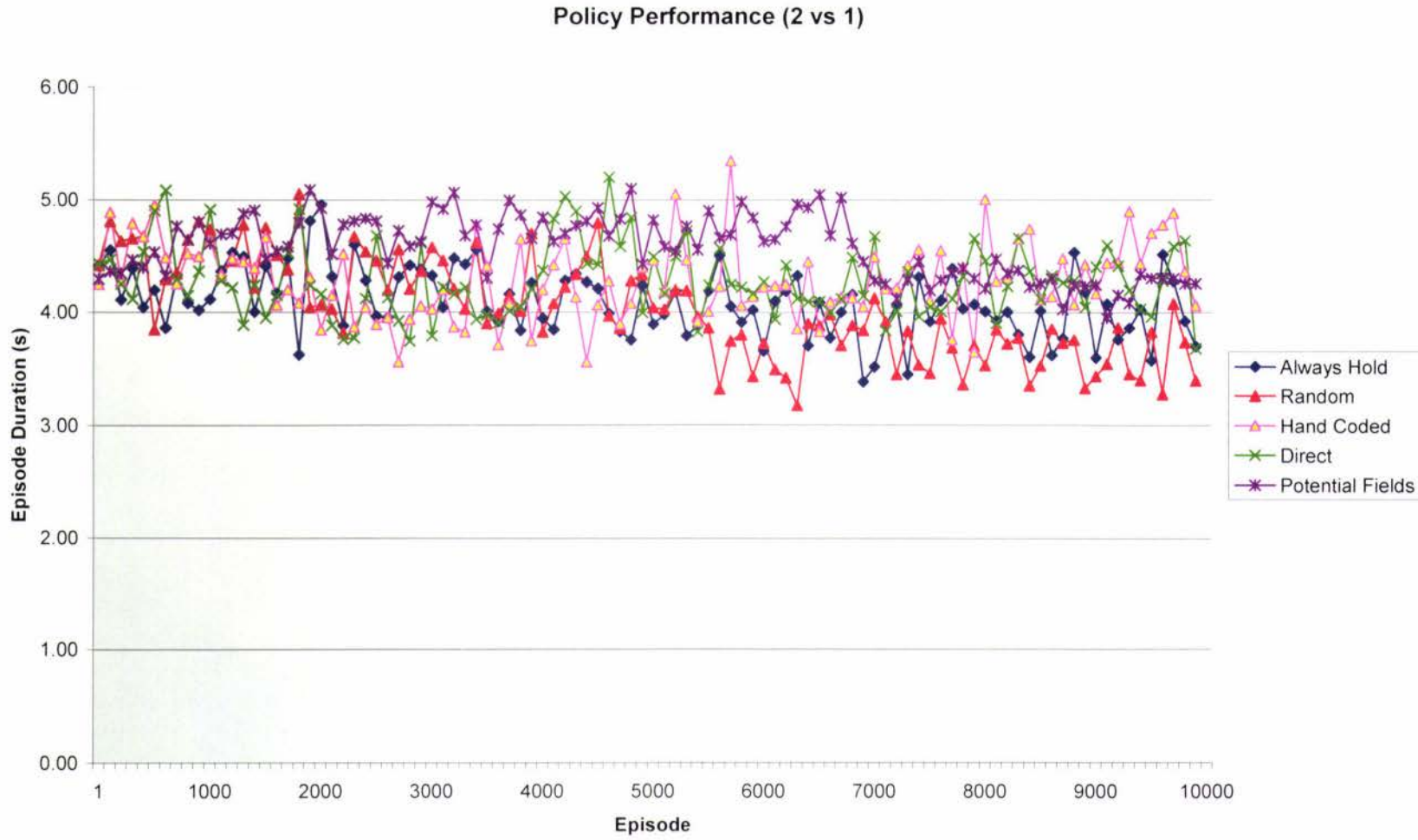
```
                // Radius of taker field is the distance from keeper to
                opponent * confidence level

                radius = state[index+j] *
                                tOrdered[j].stats[INTERCEPT].confidence;

                takerFields[j].setCircle(centre, radius);
        }

        // Reduce the radius of any keeper field that overlaps with a
        takers
        for(int j=0; j<numK; j++) {
                for(int i=0; i<numT; i++) {
                        overlap =
                        keeperFields[j].getIntersectionArea(takerFields[i]);

                        if(overlap > 0.0) {
                                area = keeperFields[j].getArea() - overlap;
                                radius = sqrt(area/M_PI);
                                keeperFields[j].setRadius(radius);
                        }
                }
        }

        // Pass to the keeper whose field is the largest or hold the ball
        double maxField = -1.0;

        for(int j=0; j<numK; j++) {
                if(keeperFields[j].getRadius() >= maxField) {
                        maxField = keeperFields[j].getRadius();
                        action = j;
                }
        }

        return action;
}

// ------------------------------------------------------------------------
// Procedure: Update
// Purpose:   Updates the Confidence Models based on the outcome of the previous
//            action.
// Params In: state - array of doubles representing current state.
//            action - integer representing action taken.
//            reward - double representing outcome of previous action, either
//                     positive or negative.
// Params Out: None.
// ------------------------------------------------------------------------
void LearningAgent::update(double state[], int action, double reward)
{
        int id;

        // If episode went in Keepers favour adjust models accordingly
        if(reward > 0.0) {

                // Adjust models based on successful kick action
                if(action > 0) {

                        // Include current player's successful pass
                        id = kOrdered[0].myID;
                        keepers[id].stats[PASS].successes++;
                        keepers[id].stats[PASS].total++;
```

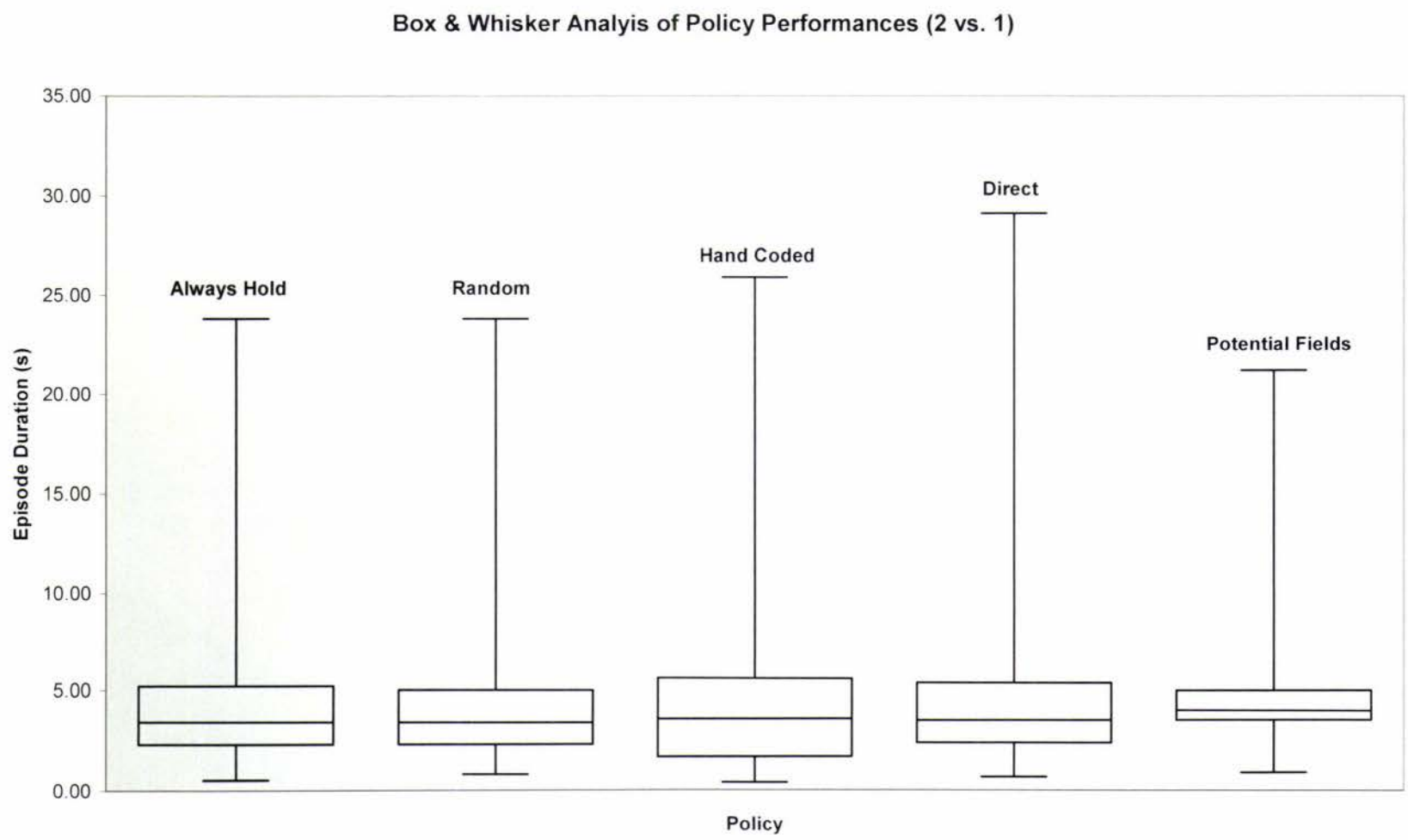
```
                keepers[id].stats[PASS].confidence = (double)
                keepers[id].stats[PASS].successes /
                keepers[id].stats[PASS].total;

                // Include target keepers successful intercept
                id = kOrdered[action].myID;
                keepers[id].stats[INTERCEPT].successes++;
                keepers[id].stats[INTERCEPT].total++;
                keepers[id].stats[INTERCEPT].confidence = (double)
                keepers[id].stats[INTERCEPT].successes /
                keepers[id].stats[INTERCEPT].total;

                // Include closest takers failed intercept
                id = tOrdered[0].myID;
                takers[id].stats[INTERCEPT].total++;
                takers[id].stats[INTERCEPT].confidence = (double)
                takers[id].stats[INTERCEPT].successes /
                takers[id].stats[INTERCEPT].total;

        // Adjust models based on successful hold action
        } else {

                // Include current player's successful hold
                id = kOrdered[0].myID;
                keepers[id].stats[EVADE].successes++;
                keepers[id].stats[EVADE].total++;
                keepers[id].stats[EVADE].confidence = (double)
                keepers[id].stats[EVADE].successes /
                keepers[id].stats[EVADE].total;

                // Include takers unsuccessful mark
                id = tOrdered[0].myID;
                takers[id].stats[MARK].total++;
                takers[id].stats[MARK].confidence = (double)
                takers[id].stats[MARK].successes /
                takers[id].stats[MARK].total;
        }

// If episode went in favour of the takers
} else {

        // Adjust models based on unsuccessful kick action
        if(action > 0) {

                // Include current player's unsuccessful pass
                id = kOrdered[0].myID;
                keepers[id].stats[PASS].total++;
                keepers[id].stats[PASS].confidence = (double)
                keepers[id].stats[PASS].successes /
                keepers[id].stats[PASS].total;

                // Include target keepers unsuccessful intercept
                id = kOrdered[action].myID;
                keepers[id].stats[INTERCEPT].total++;
                keepers[id].stats[INTERCEPT].confidence = (double)
                keepers[id].stats[INTERCEPT].successes /
                keepers[id].stats[INTERCEPT].total;

                // Include takers successful intercept
                id = tOrdered[0].myID;
                takers[id].stats[INTERCEPT].successes++;
                takers[id].stats[INTERCEPT].total++;
                takers[id].stats[INTERCEPT].confidence = (double)
```

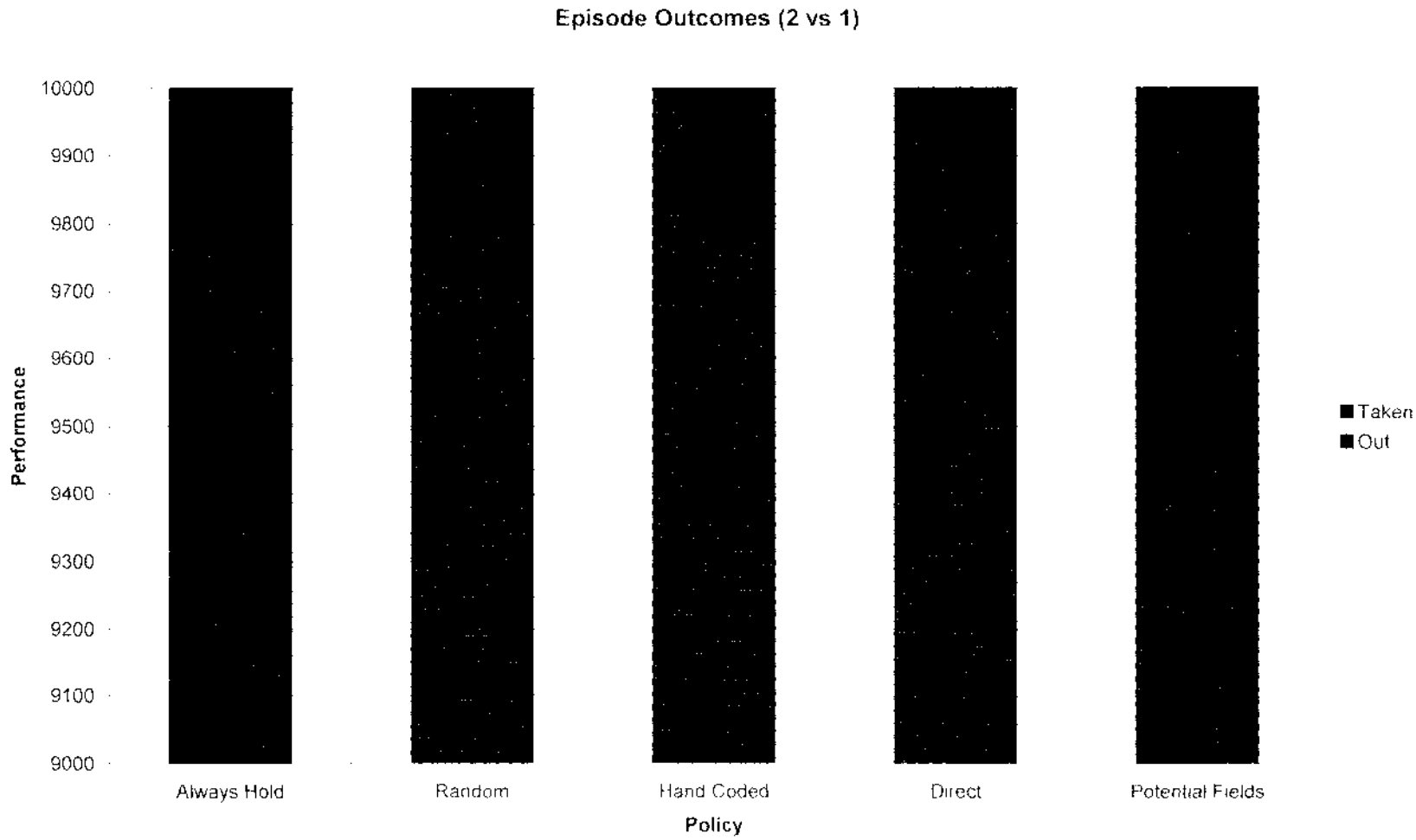
```
                takers[id].stats[INTERCEPT].successes /
                takers[id].stats[INTERCEPT].total;

            // Adjust models based on unsuccessful hold action
            } else {

                // Include current players's unsuccessful evade
                id = kOrdered[0].myID;
                keepers[id].stats[EVADE].total++;
                keepers[id].stats[EVADE].confidence = (double)
                keepers[id].stats[EVADE].successes /
                keepers[id].stats[EVADE].total;

                // Include takers successful intercept
                id = tOrdered[0].myID;
                takers[id].stats[MARK].successes++;
                takers[id].stats[MARK].total++;
                takers[id].stats[MARK].confidence = (double)
                takers[id].stats[MARK].successes /
                takers[id].stats[MARK].total;
            }
        }
}
```

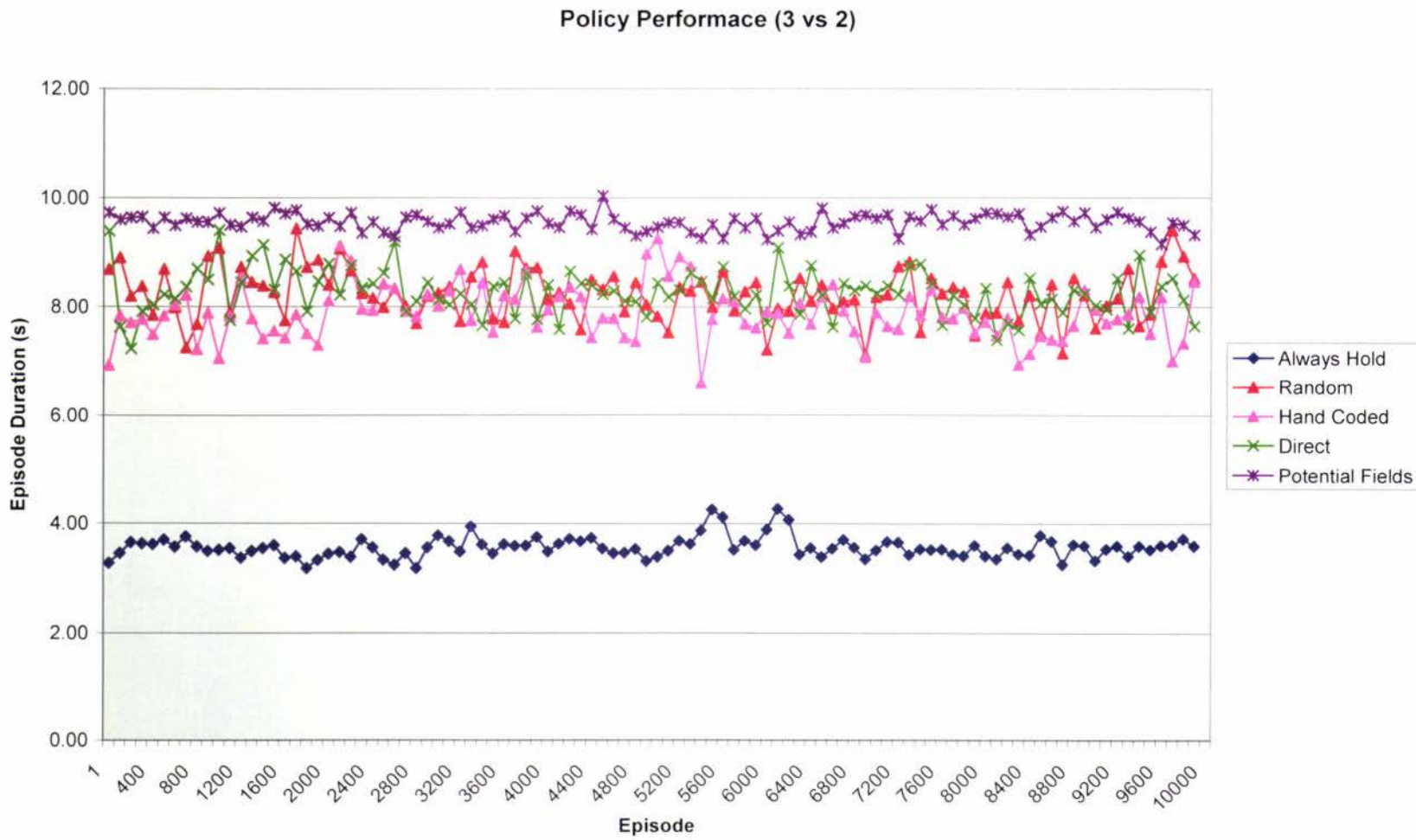
## Appendix A2: Result Graphs

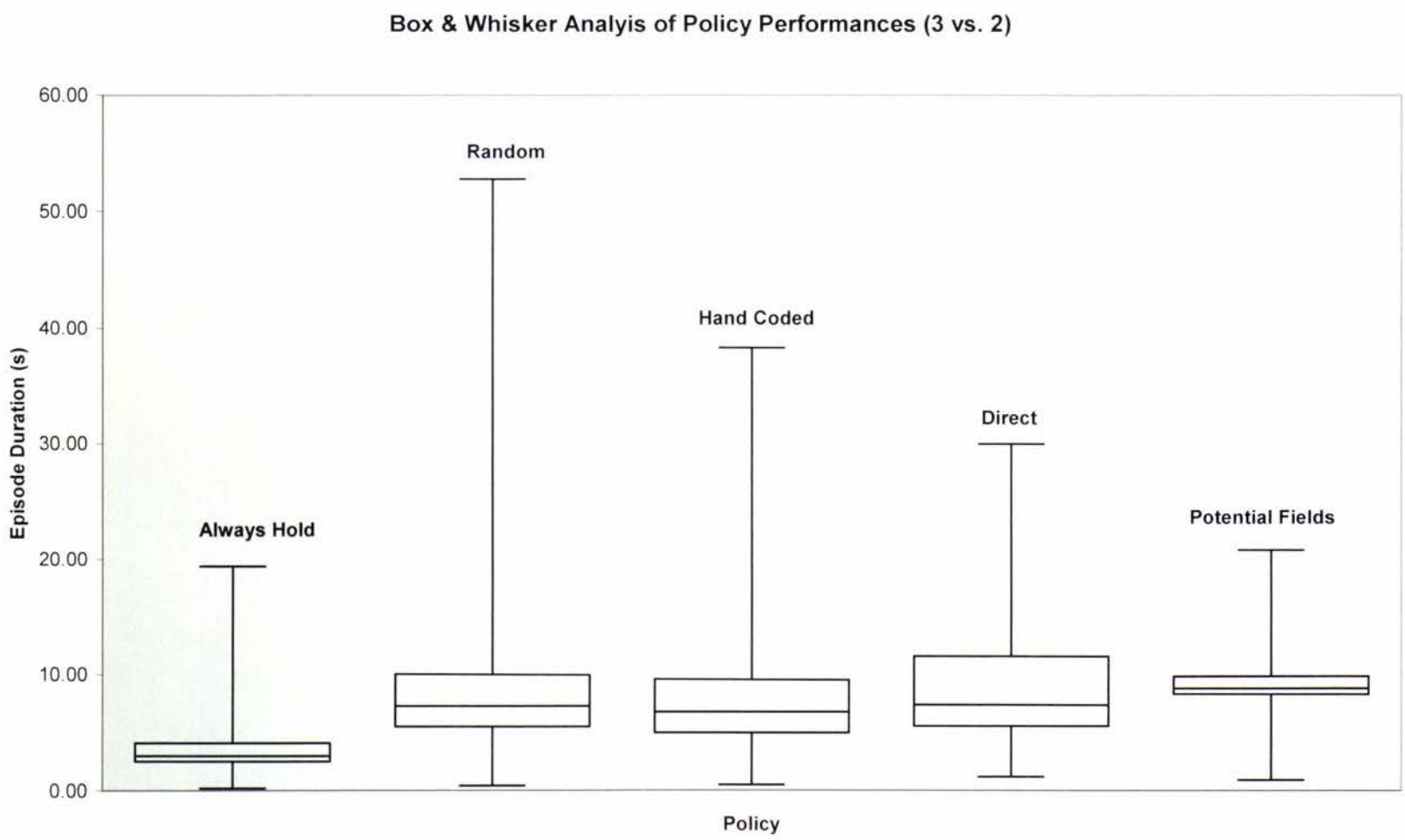
## A2.1: Feasibility Testing Graphs

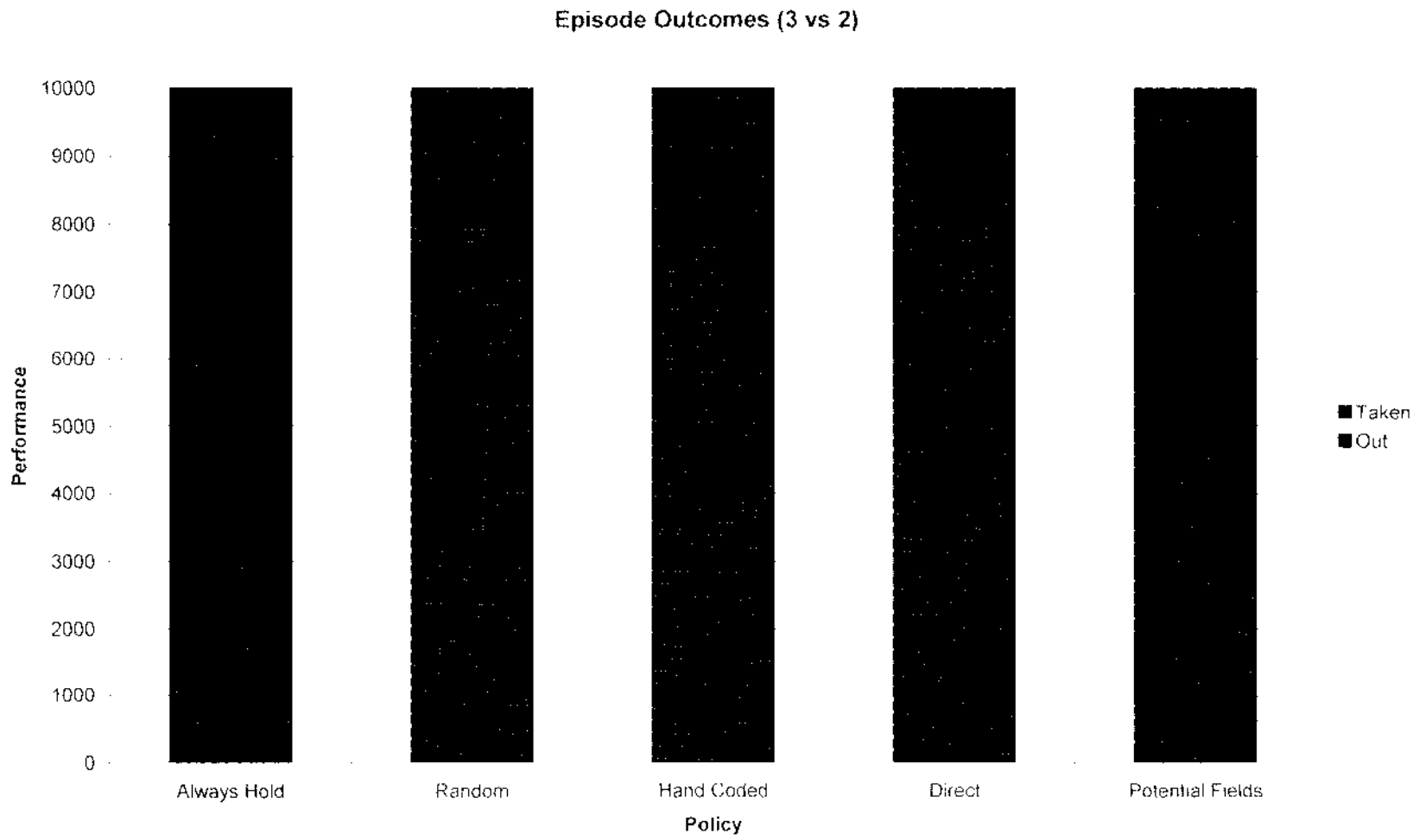


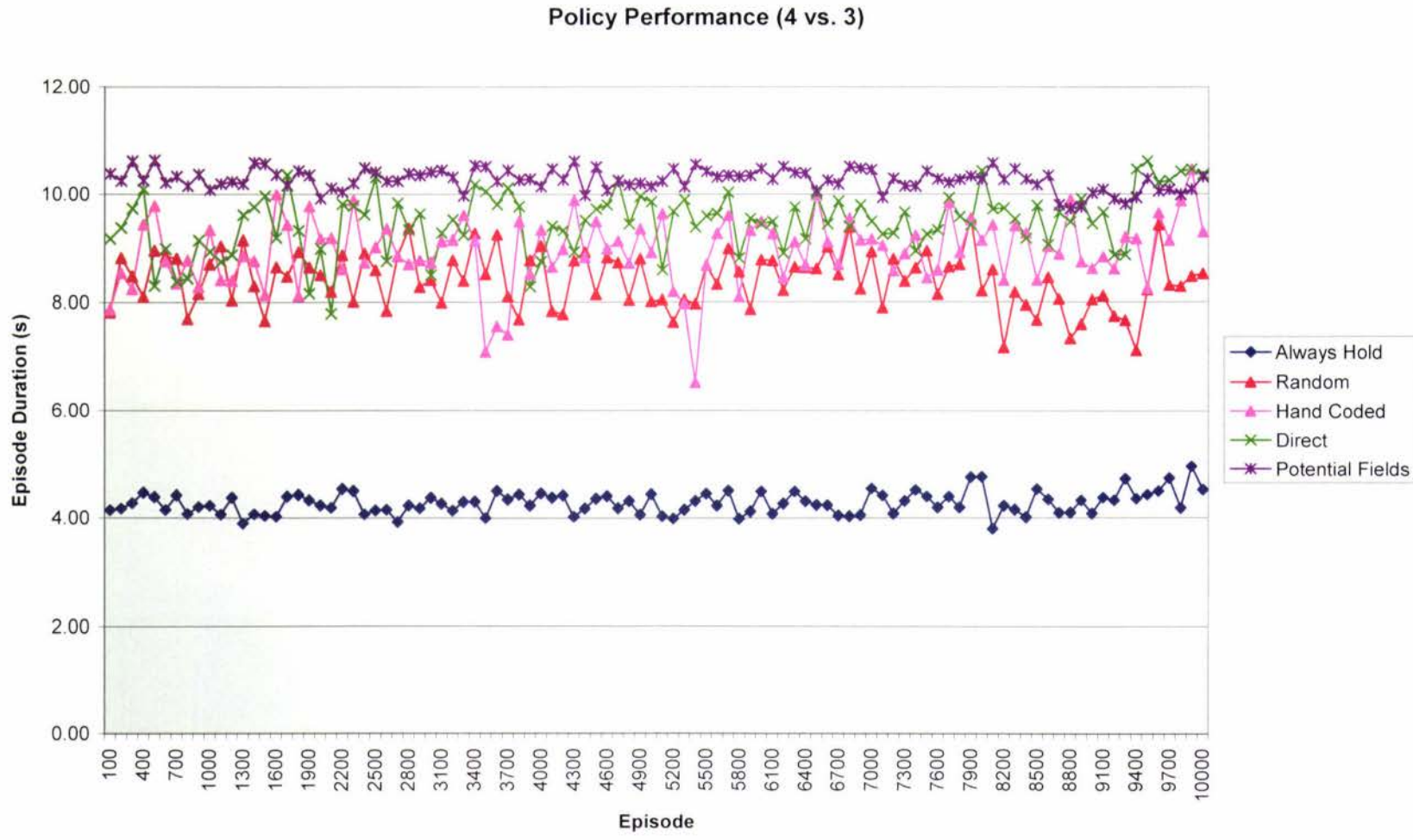

Box & Whisker Analyis of Policy Performances (2 vs. 1)
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
Episode Duration (s)
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy

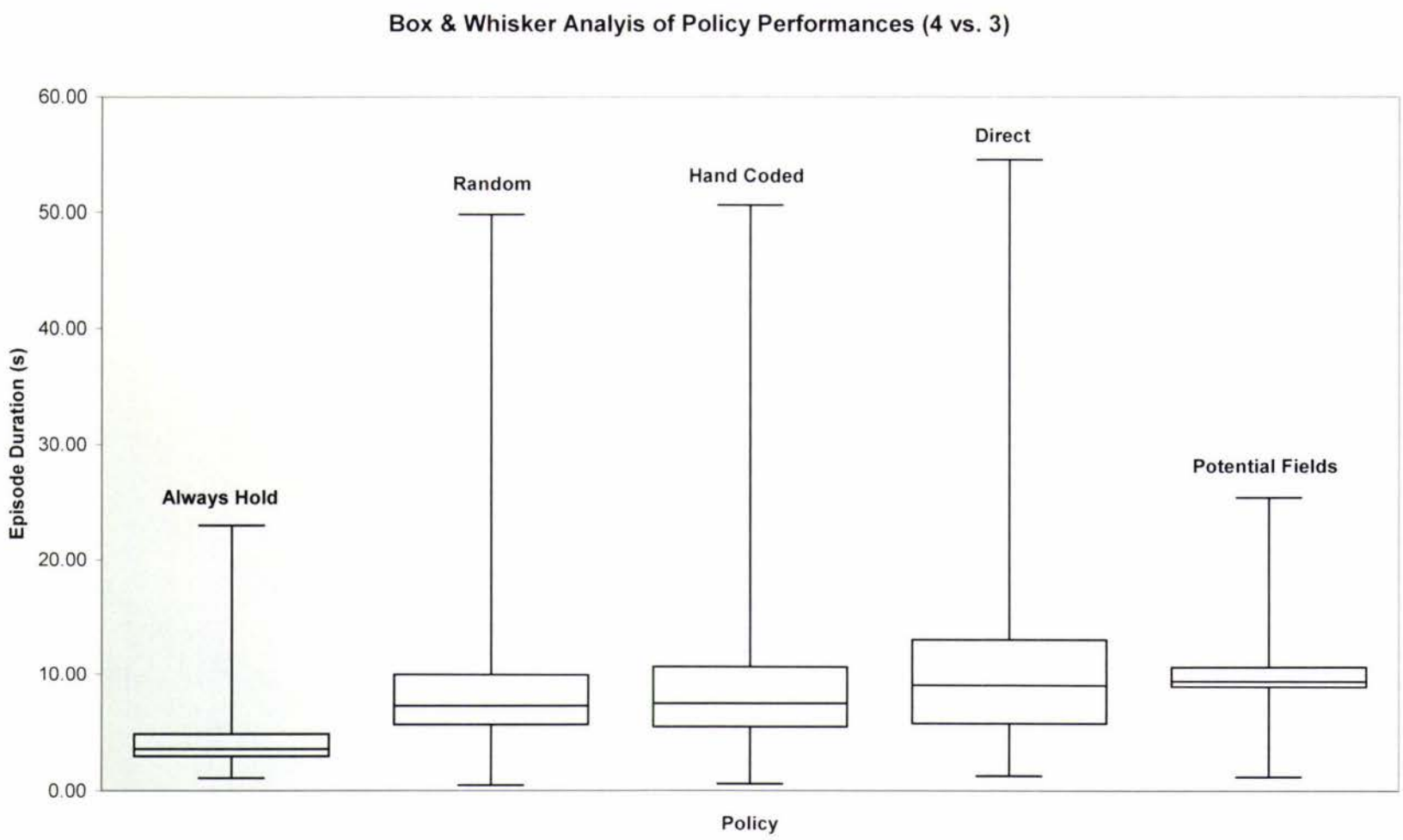

Episode Outcomes (2 vs 1)
10000
9900
9800
9700
9600
9500
9400
9300
9200
9100
9000
Performance
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy
Taken
Out

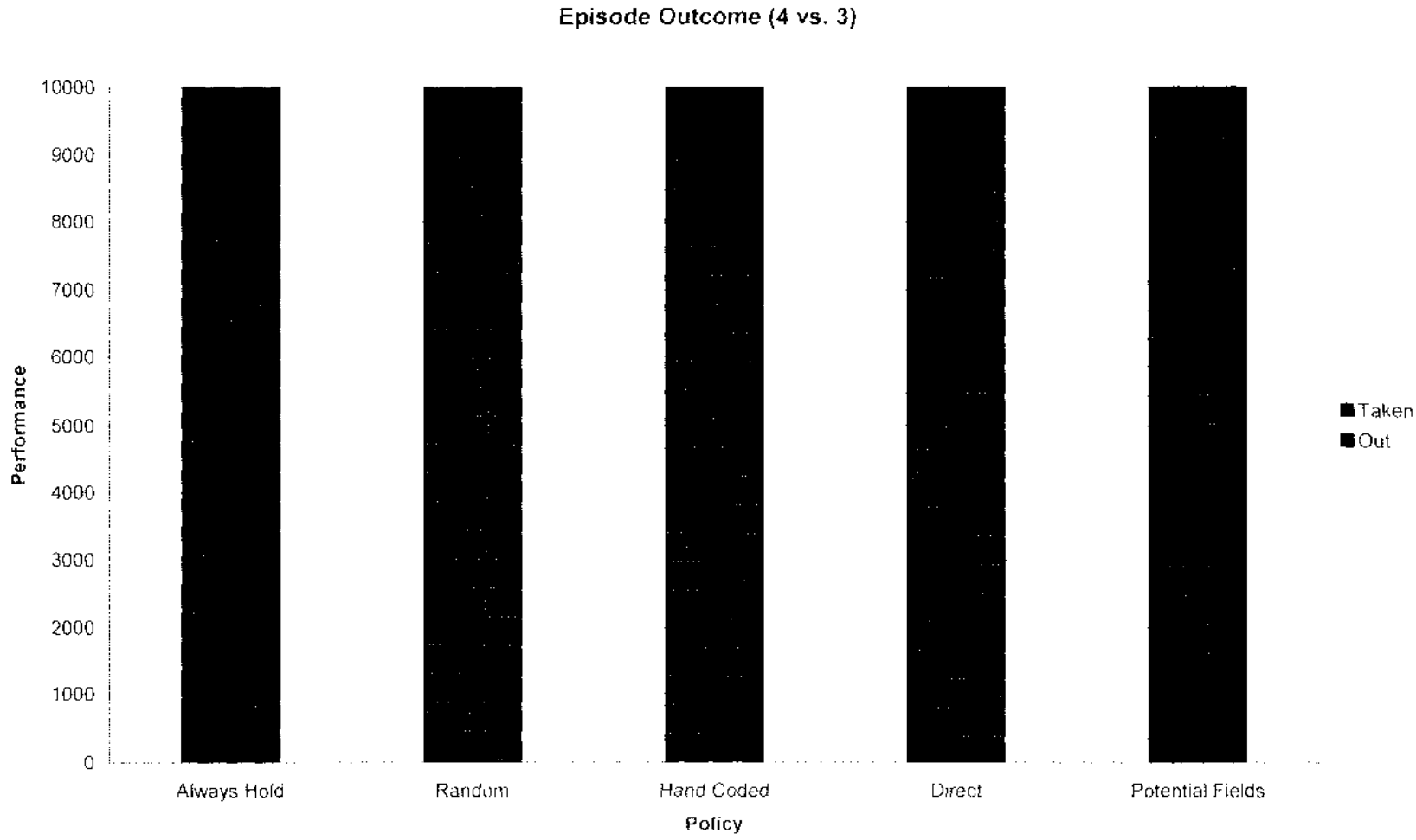

Policy Performace (3 vs 2)
Episode Duration (s)
12.00
10.00
8.00
6.00
4.00
2.00
0.00
1
400
800
1200
1600
2000
2400
2800
3200
3600
4000
4400
4800
5200
5600
6000
6400
6800
7200
7600
8000
8400
8800
9200
9600
10000
Episode
Always Hold
Random
Hand Coded
Direct
Potential Fields

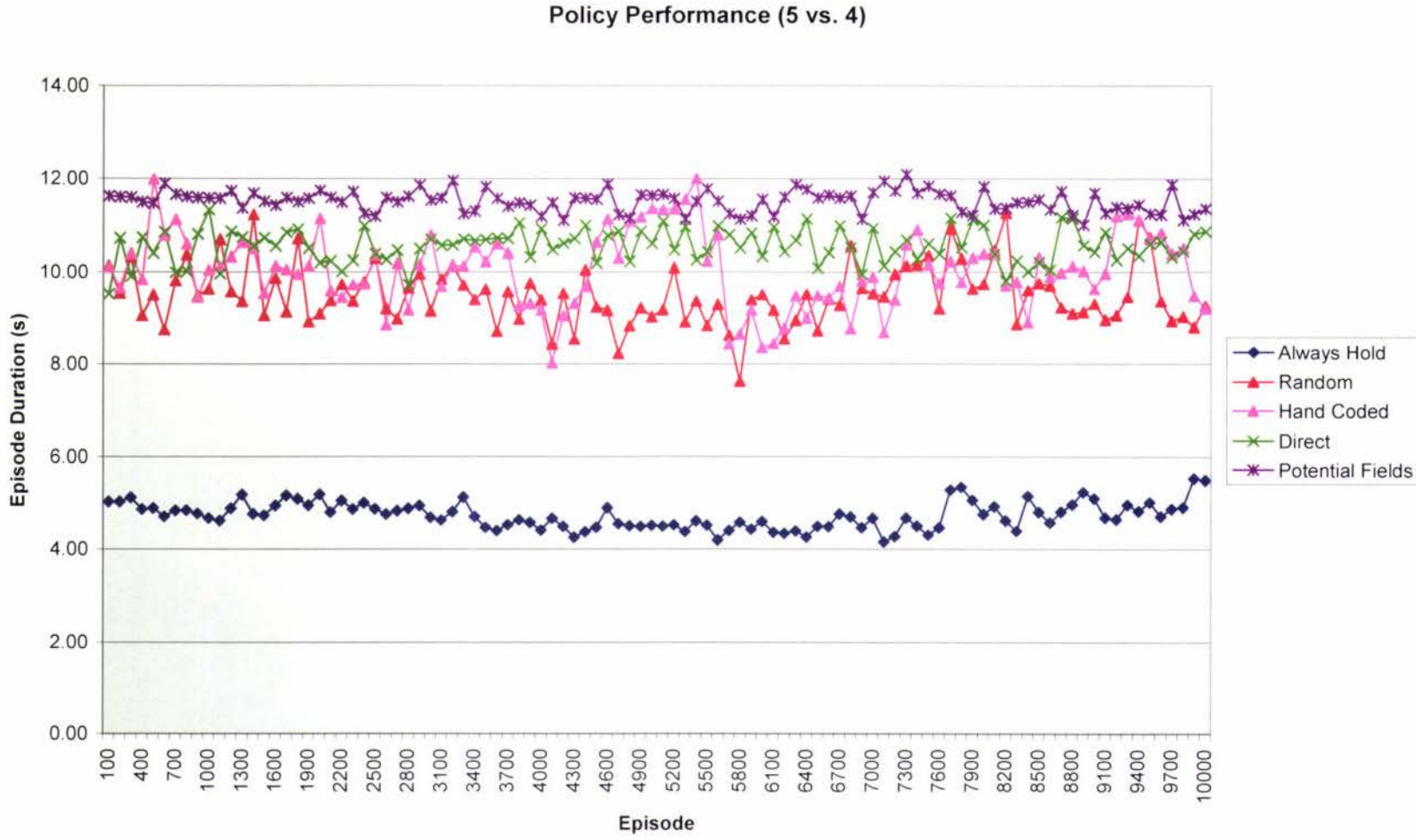

Box & Whisker Analyis of Policy Performances (3 vs. 2)
60.00
50.00
40.00
30.00
20.00
10.00
0.00
Episode Duration (s)
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy

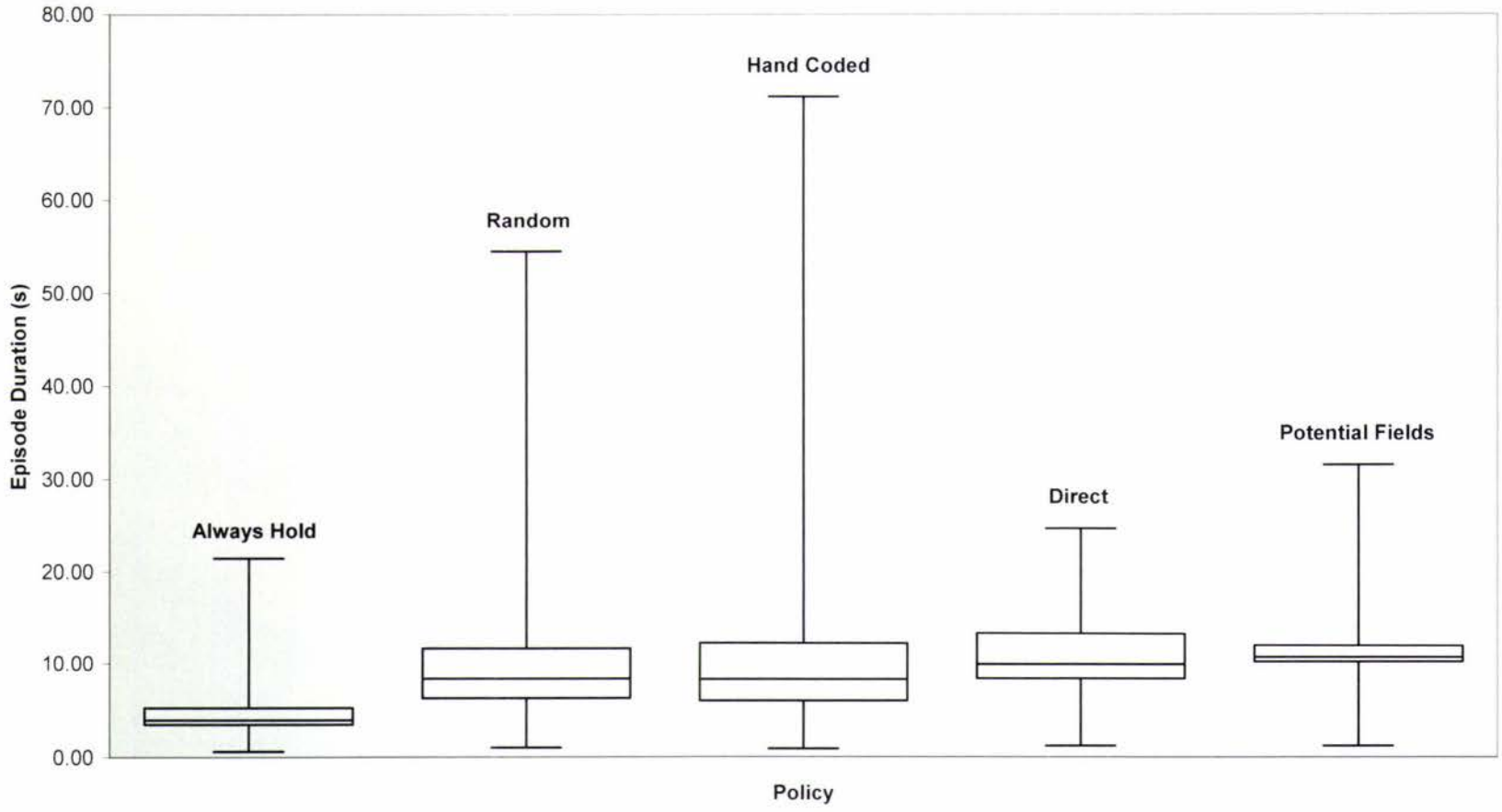

Episode Outcomes (3 vs 2)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Performance
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy
Taken
Out

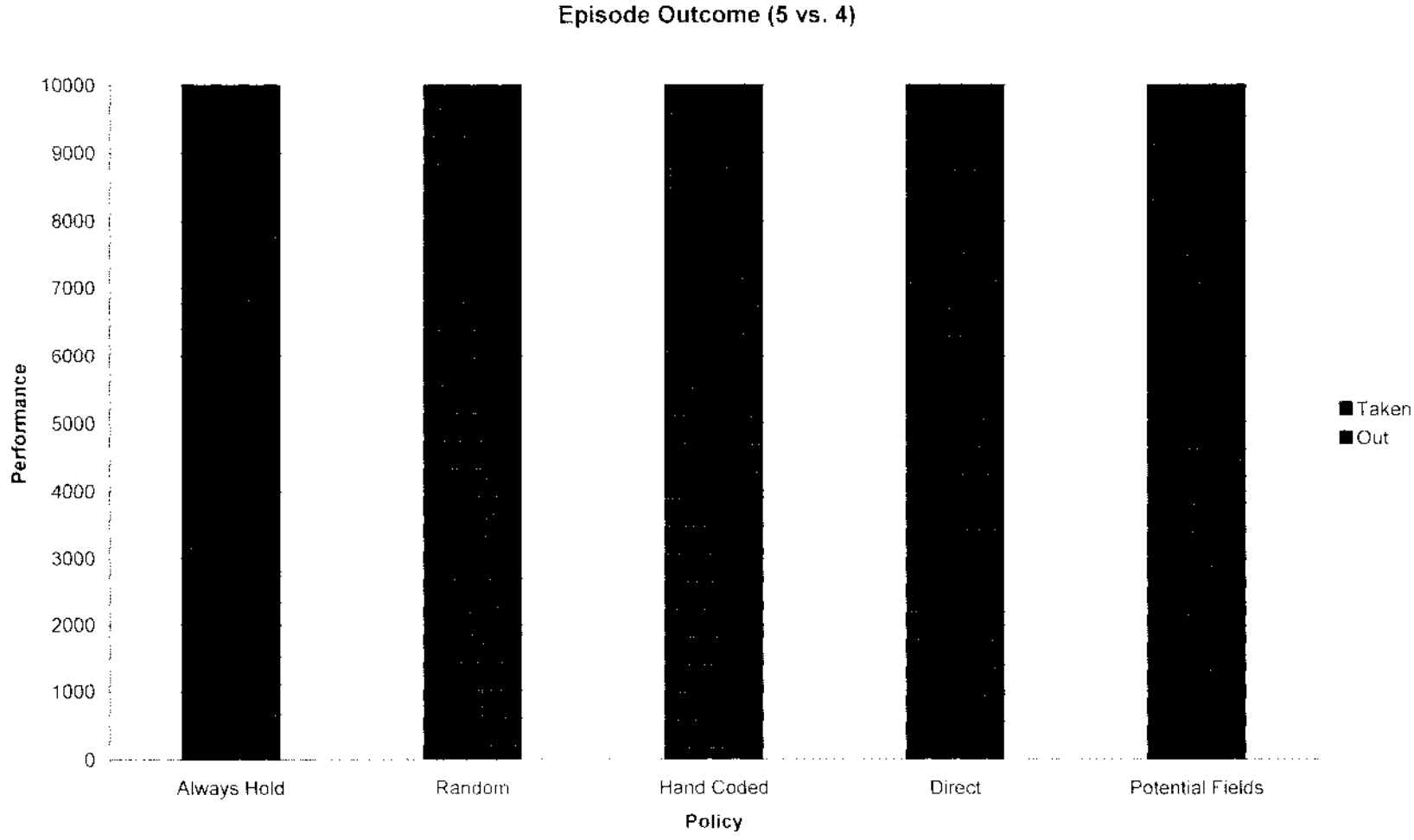

Policy Performance (4 vs. 3)
Episode Duration (s)
12.00
10.00
8.00
6.00
4.00
2.00
0.00
100
400
700
1000
1300
1600
1900
2200
2500
2800
3100
3400
3700
4000
4300
4600
4900
5200
5500
5800
6100
6400
6700
7000
7300
7600
7900
8200
8500
8800
9100
9400
9700
10000
Episode
Always Hold
Random
Hand Coded
Direct
Potential Fields


Box & Whisker Analyis of Policy Performances (4 vs. 3)
60.00
50.00
40.00
30.00
20.00
10.00
0.00
Episode Duration (s)
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy

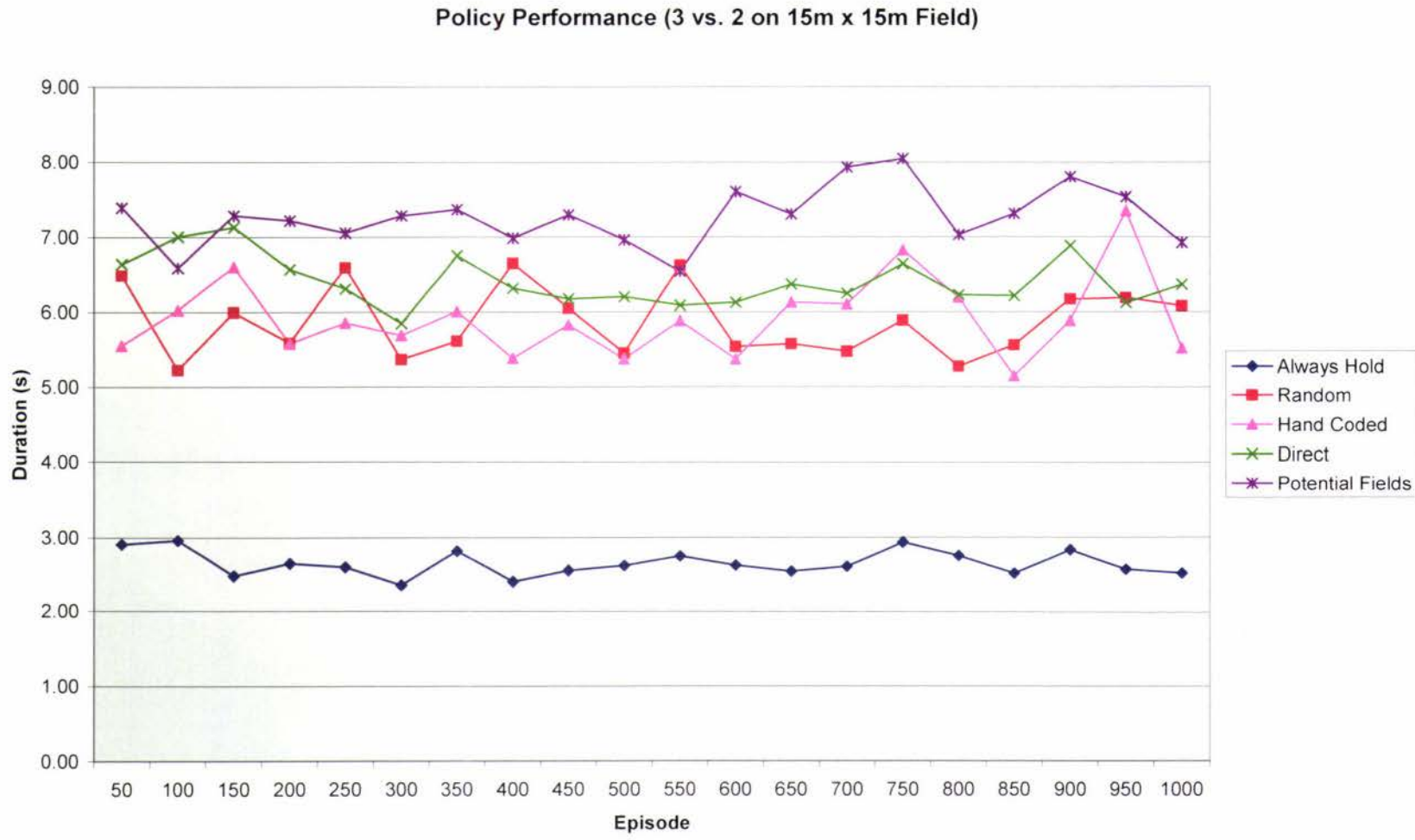

Episode Outcome (4 vs. 3)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Performance
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy
Taken
Out

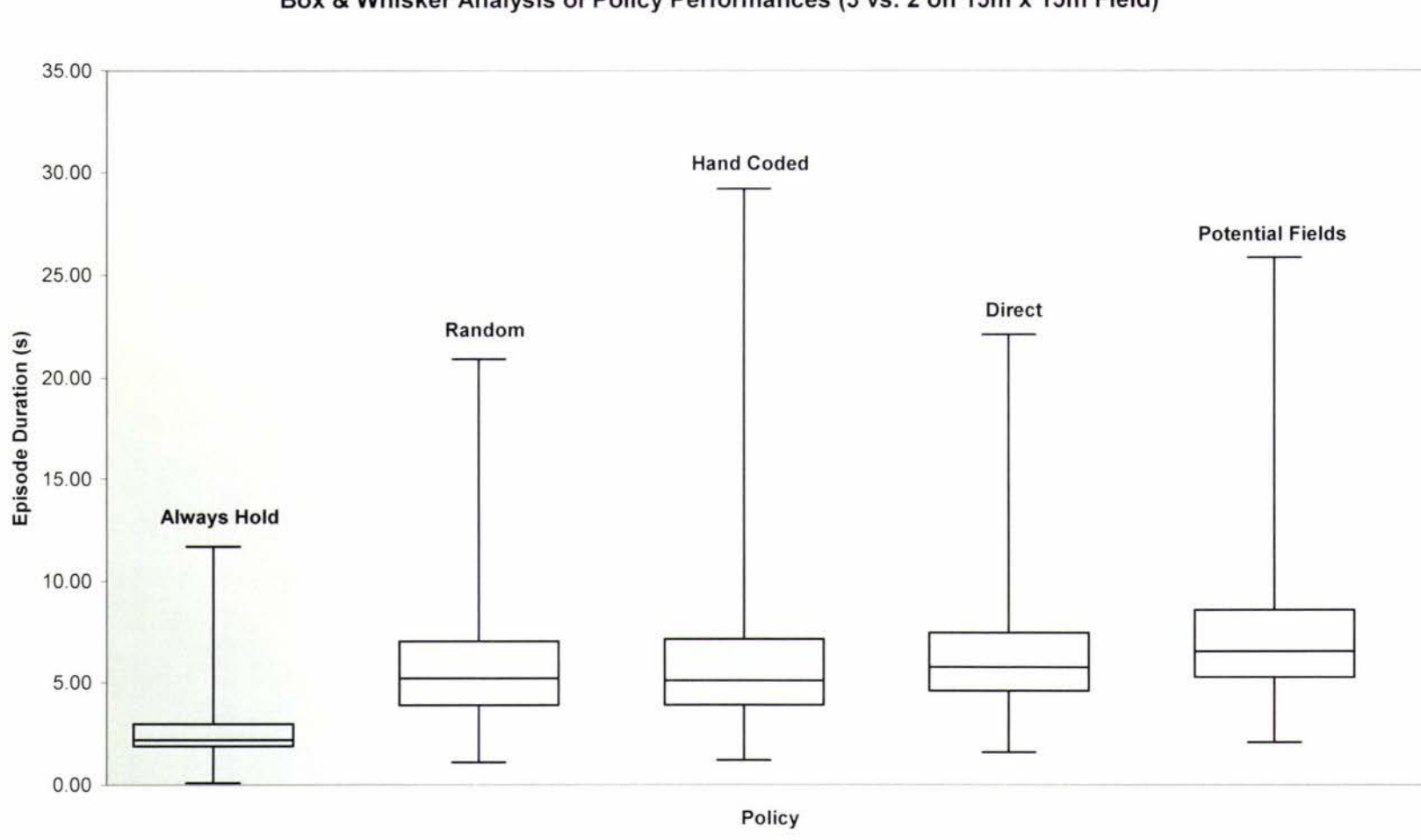

Policy Performance (5 vs. 4)
Episode Duration (s)
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0.00
100
400
700
1000
1300
1600
1900
2200
2500
2800
3100
3400
3700
4000
4300
4600
4900
5200
5500
5800
6100
6400
6700
7000
7300
7600
7900
8200
8500
8800
9100
9400
9700
10000
Episode
Always Hold
Random
Hand Coded
Direct
Potential Fields

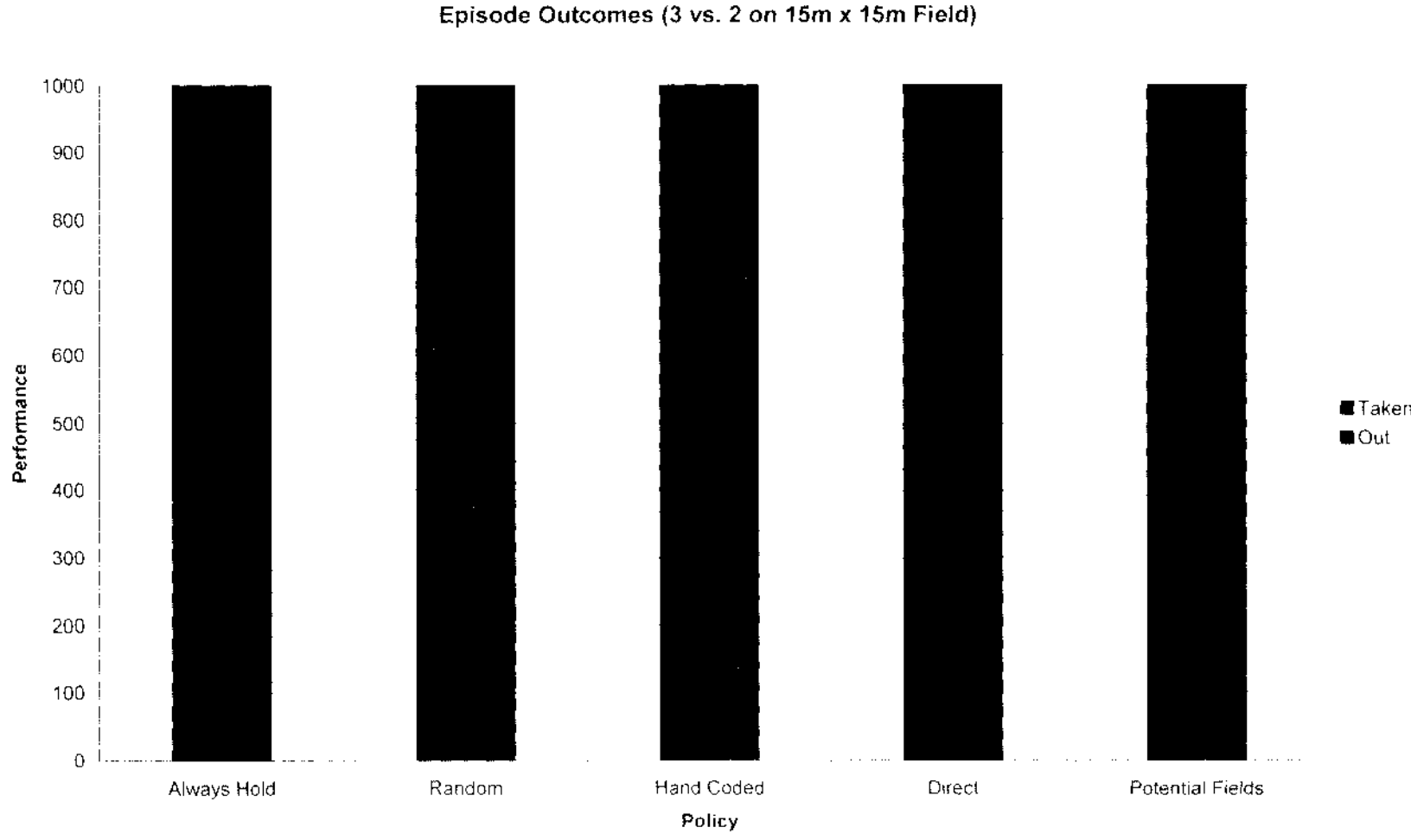

Box & Whisker Analyis of Policy Performances (5 vs. 4)
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
Episode Duration (s)
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy

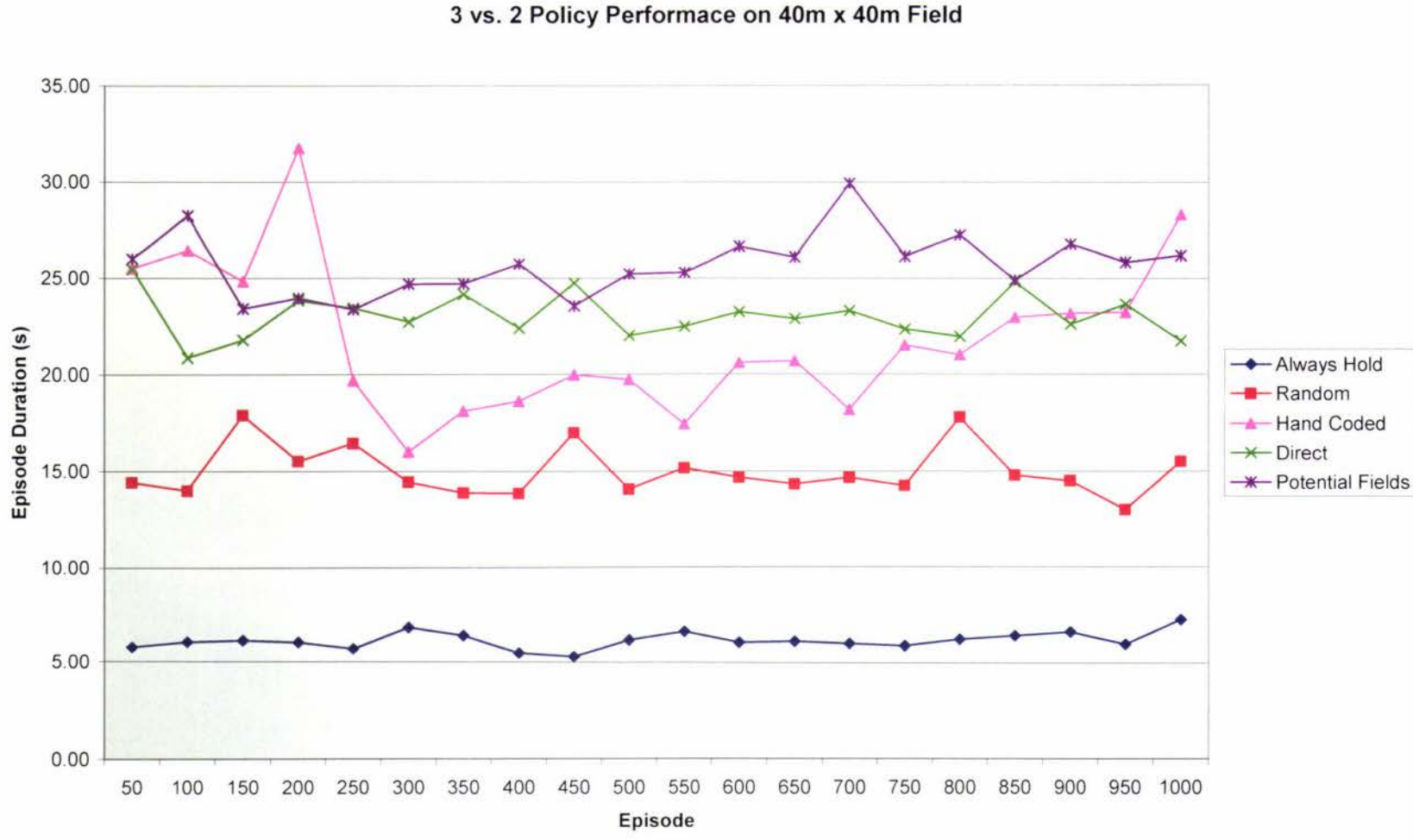

Episode Outcome (5 vs. 4)
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Performance
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy
Taken
Out

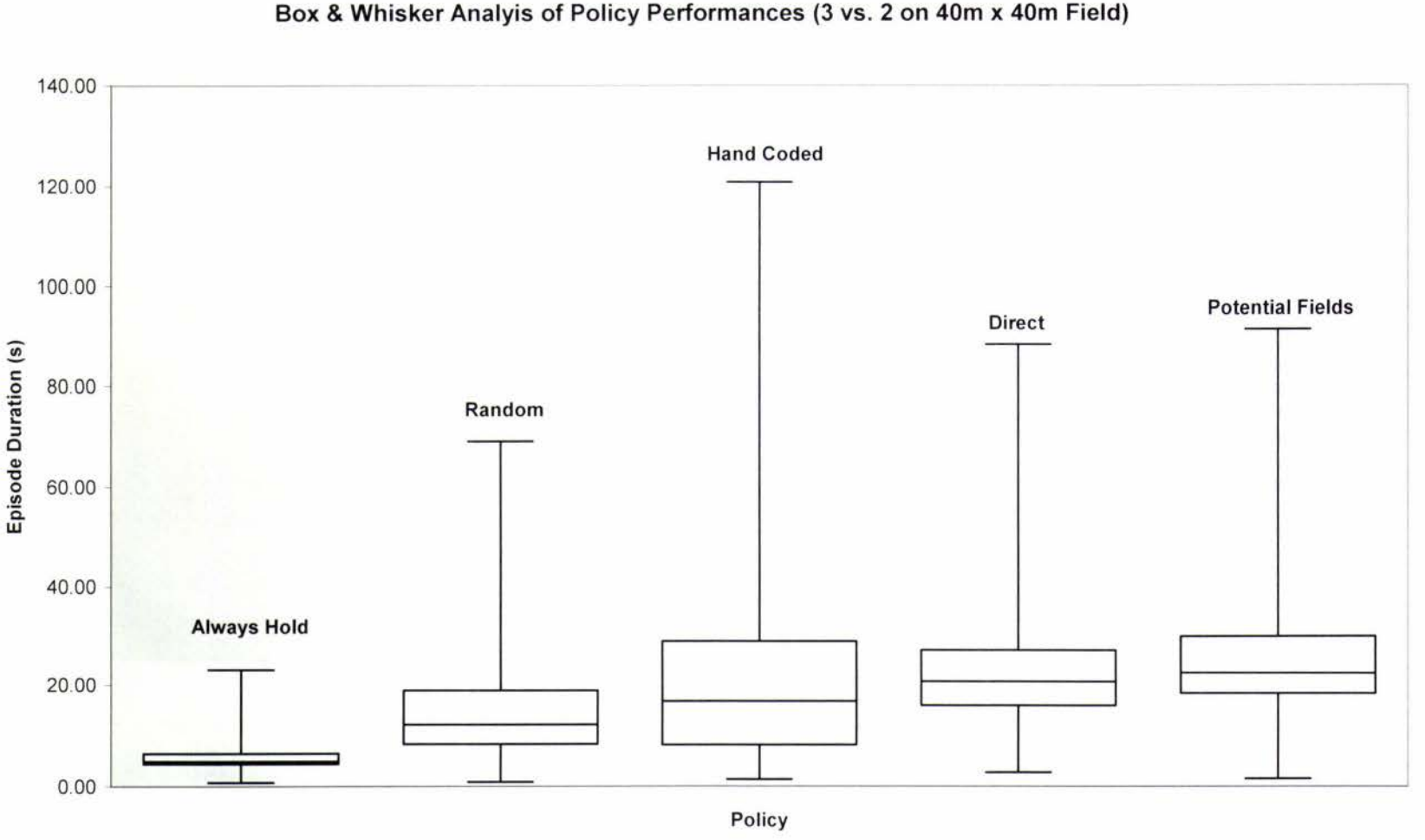
## A2.2: Scalability Testing Graphs

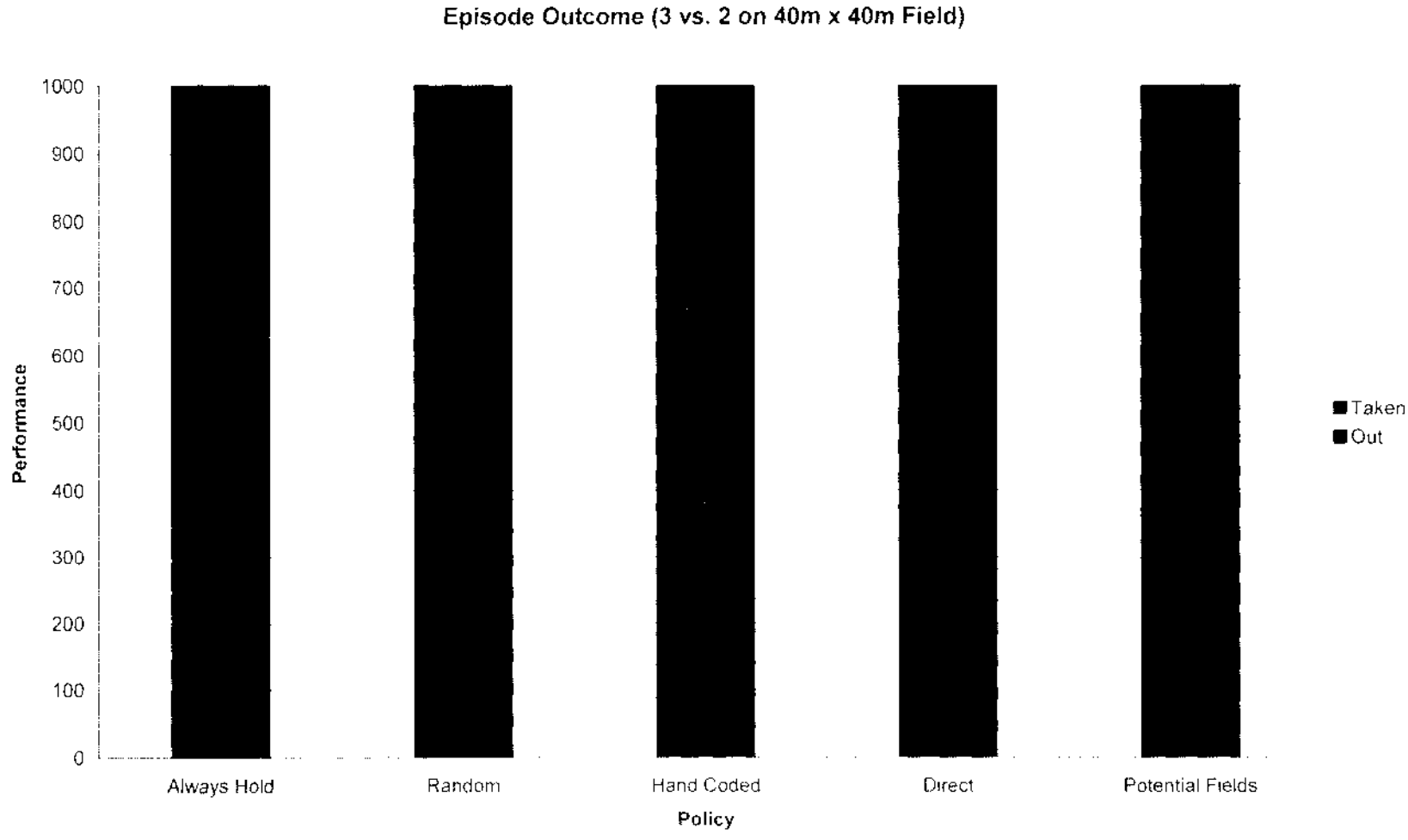


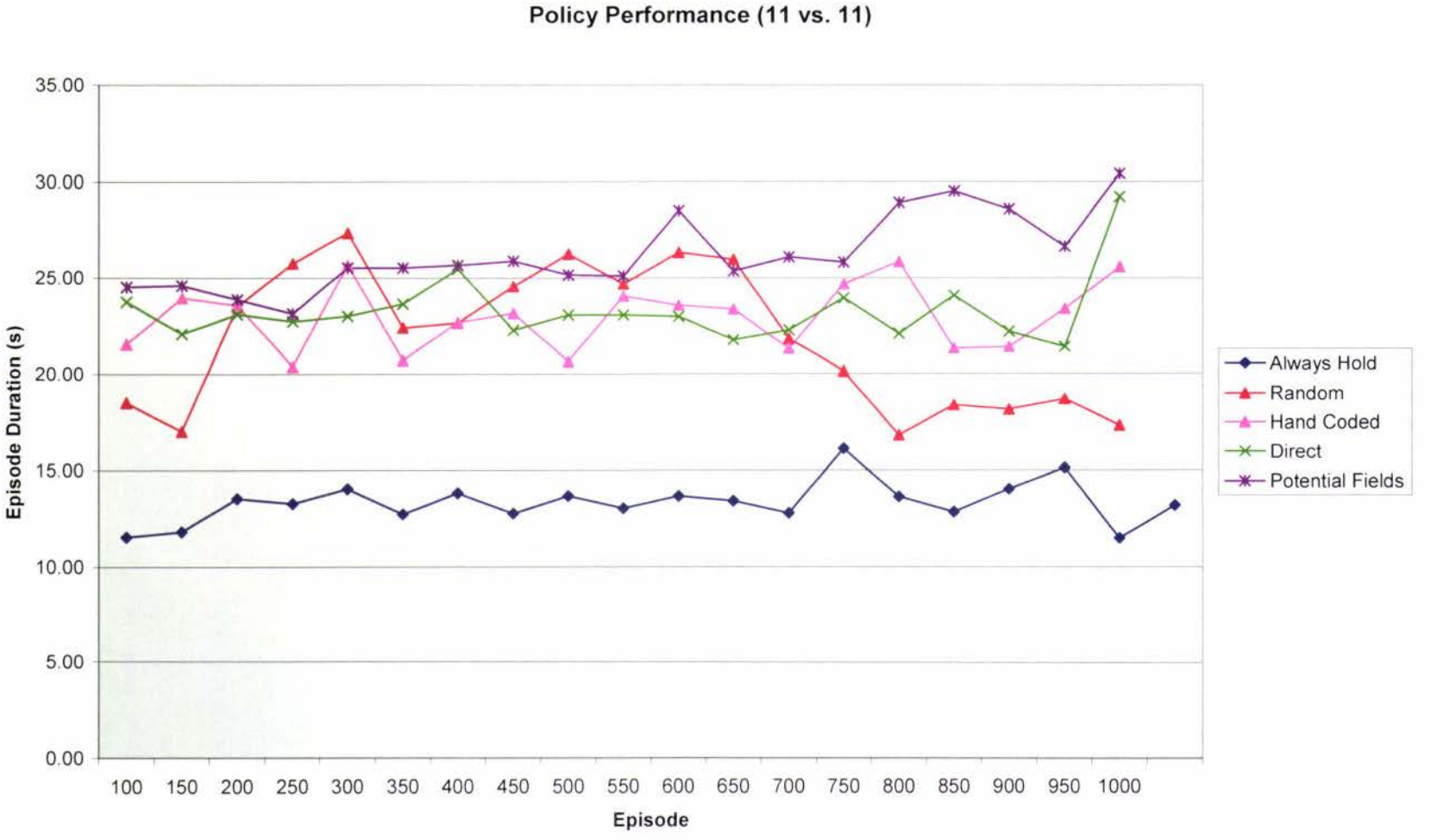

Box & Whisker Analysis of Policy Performances (3 vs. 2 on 15m x 15m Field)
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
Episode Duration (s)
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy

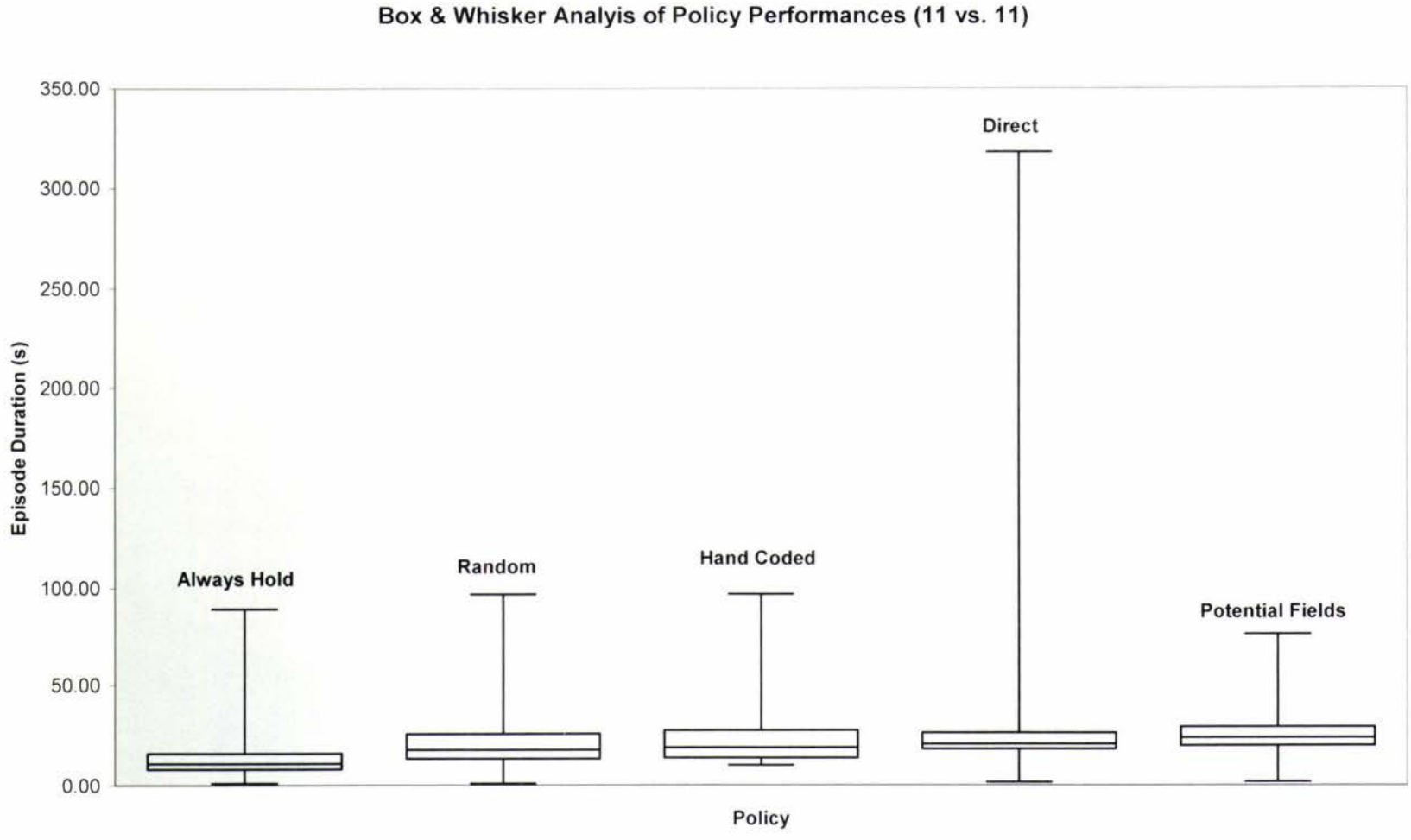

Episode Outcomes (3 vs. 2 on 15m x 15m Field)
Performance
1000
900
800
700
600
500
400
300
200
100
0
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy
Taken
Out

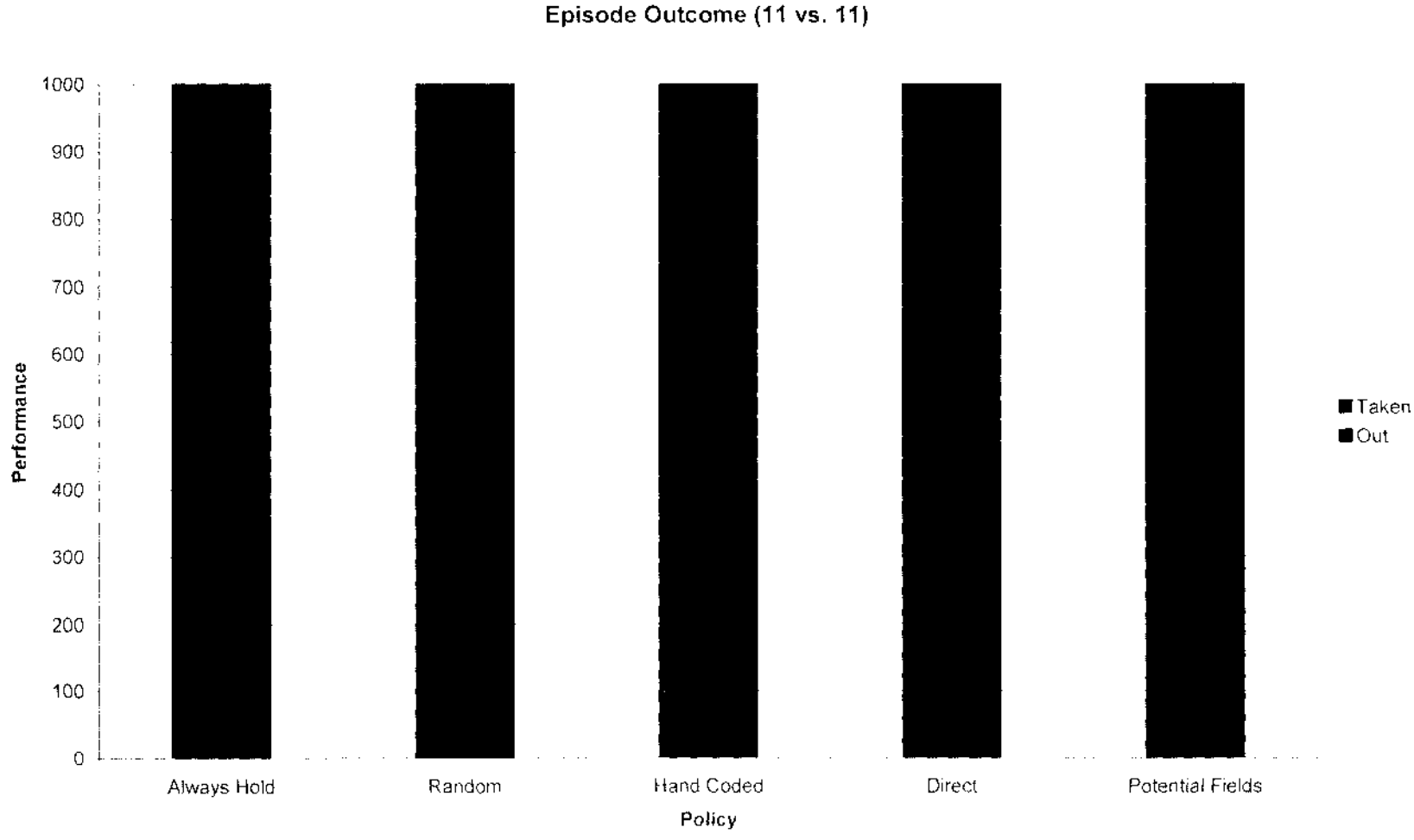

3 vs. 2 Policy Performace on 40m x 40m Field
Episode Duration (s)
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000
Episode
Always Hold
Random
Hand Coded
Direct
Potential Fields


Box & Whisker Analyis of Policy Performances (3 vs. 2 on 40m x 40m Field)
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
Episode Duration (s)
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy

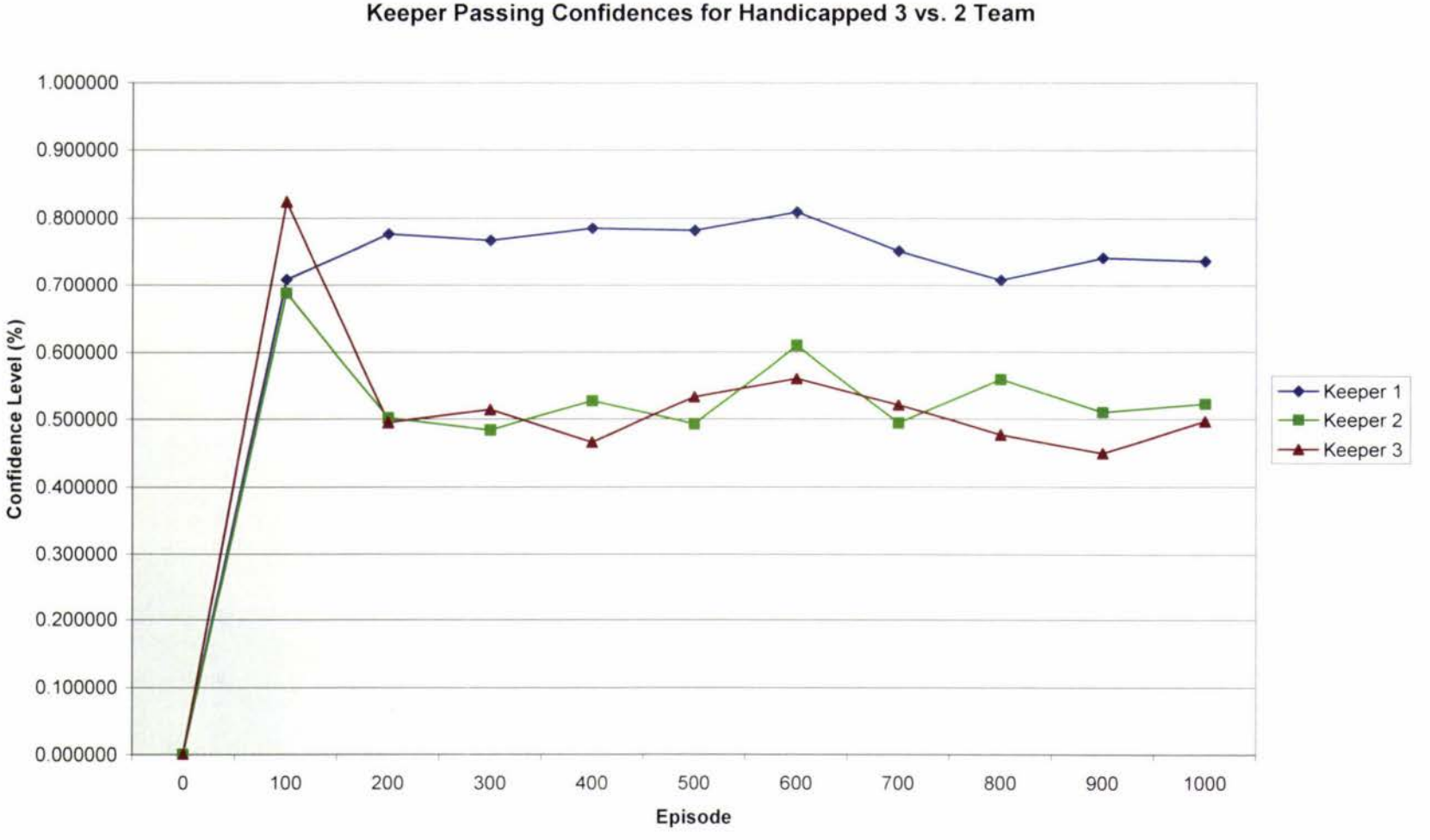

Episode Outcome (3 vs. 2 on 40m x 40m Field)
Performance
1000
900
800
700
600
500
400
300
200
100
0
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy
Taken
Out

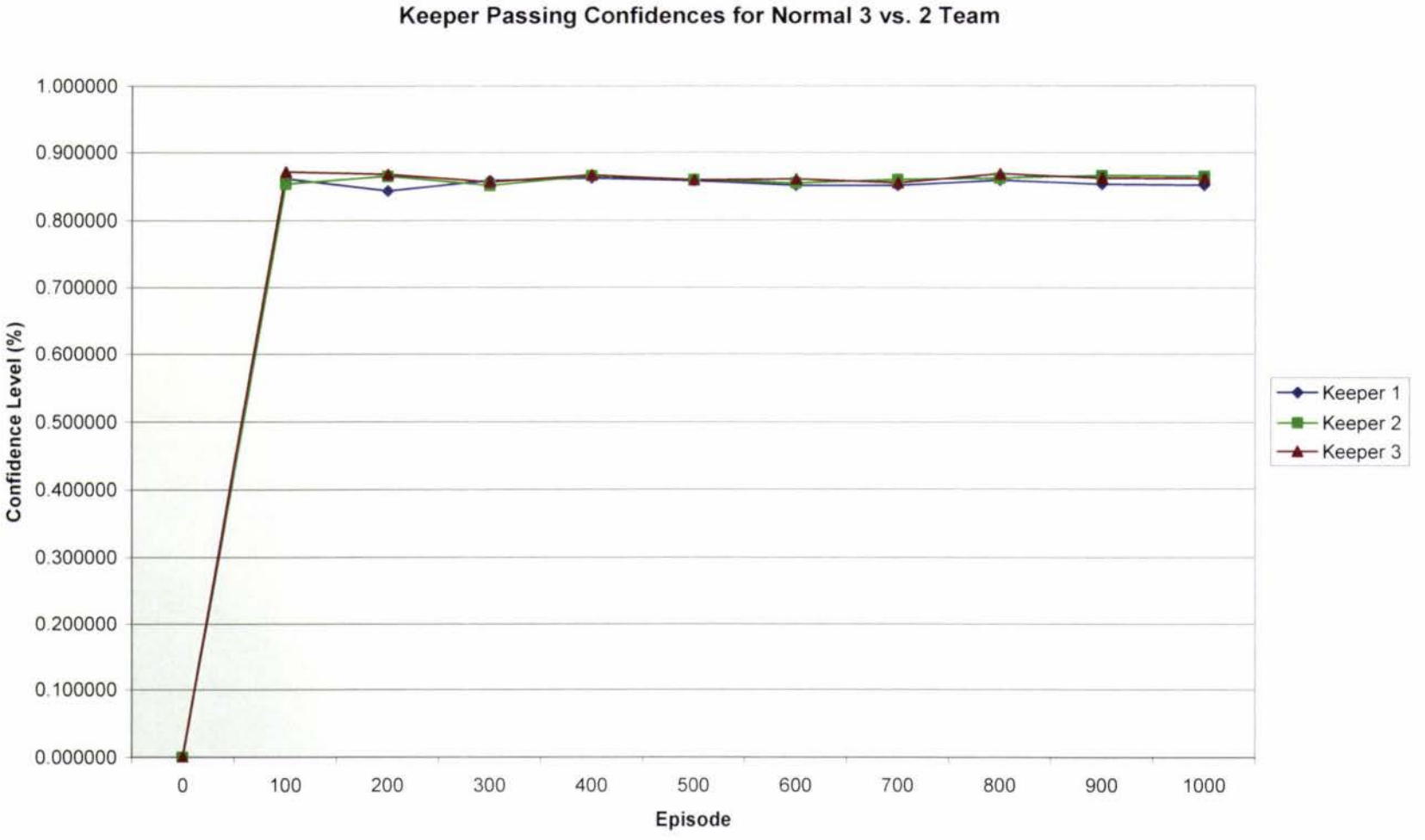

Policy Performance (11 vs. 11)
Episode Duration (s)
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950 1000
Episode
Always Hold
Random
Hand Coded
Direct
Potential Fields


Box & Whisker Analyis of Policy Performances (11 vs. 11)
350.00
300.00
250.00
200.00
150.00
100.00
50.00
0.00
Episode Duration (s)
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy


Episode Outcome (11 vs. 11)
Performance
1000
900
800
700
600
500
400
300
200
100
0
Always Hold
Random
Hand Coded
Direct
Potential Fields
Policy
Taken
Out

## A2.3: Entropy Testing Graphs




Keeper Passing Confidences for Normal 3 vs. 2 Team
1.000000
0.900000
0.800000
0.700000
0.600000
0.500000
0.400000
0.300000
0.200000
0.100000
0.000000
Confidence Level (%)
0
100
200
300
400
500
600
700
800
900
1000
Episode
Keeper 1
Keeper 2
Keeper 3