AFRL-IF-RS-TR-2006-116
In-House Interim Report
March 2006

ASYNCHRONOUS CHESS: A REAL-TIME, ADVERSARIAL RESEARCH ENVIRONMENT

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 074-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503			
1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** MARCH 2006	**3. REPORT TYPE AND DATES COVERED** In-House Interim Jan 2005 – Mar 2006	
4. TITLE AND SUBTITLE ASYNCHRONOUS CHESS: A REAL-TIME, ADVERSARIAL RESEARCH ENVIRONMENT			**5. FUNDING NUMBERS** C - N/A PE - 62702F PR - 558S TA - HA WU - TR
6. AUTHOR(S) James Lawton, Nathaniel Gemelli, Robert Wright, Andrew Boes			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSB 525 Brooks Road Rome New York 13441-4505			**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSB 525 Brooks Road Rome New York 13441-4505			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2006-116
11. SUPPLEMENTARY NOTES AFRL Project Engineer: James Lawton/IFSB/(315) 330-4476/ James.Lawton@rl.af.mil			
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.			**12b. DISTRIBUTION CODE**
13. ABSTRACT ***(Maximum 200 Words)***			

Asynchronous Chess (AChess) is a platform for the development and evaluation of real-time adversarial agent technologies. It is a two-player game using the basic rules of chess with the modification that agents may move as many pieces as they want at any time. Modifying chess in this way creates a new robust, asynchronous, real-time game in which agents must carefully balance their time between reasoning and acting in order to out-perform their opponent. As a fast-paced adversarial game, many challenges relevant to real-word application arise which give it merit for study and use.

14. SUBJECT TERMS Adversarial Environments, Real-Time Reasoning, Intelligent Agents			**15. NUMBER OF PAGES** 40
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Abstract

We present adversarial agent research being conducted in a real-time asynchronous environment, *Asynchronous Chess* (AChess). AChess is a real-time competitive experiment platform for developing new agent technologies using single-agent reasoning and/or multi-agent cooperative strategies. We aim to provide a simplified asynchronous environment that is stochastic in its nature, but easily described from its foundations as a synchronized game. The goal is to design agent technologies that perform better in these domains than existing single- and multi-agent methods. This research applies to non-deterministic agent-based search and reasoning technologies for use in a simplified, yet still very complex, real-time environment for competitive and adaptive agents.

Contents

List of Figures

List of Tables

1 Executive Summary

In this report, we present *Asynchronous Chess* (AChess), an in-house effort involving a new adversarial, real-time multi-agent system (MAS) research environment. AChess is an extension of traditional chess, with one major difference: asynchrony. It eliminates the rule that makes chess a turn-based game. With this one extension to chess, AChess becomes a rich environment to study competitive agents in real-time.

It is important to provide decision-makers with the ability to model and reason about activities occurring in dynamic, adversarial environments. The use of these environments will be important in providing MAS-based solutions to military-critical areas such as distributed mission planning and Unmanned Autonomous Vehicles (UAVs). Therefore, it follows that the study of MAS solutions to these areas is essential in pushing the state-of-the-art.

The first step to studying the behavior of agents and multi-agent systems in an adversarial, real-time setting is to develop a research environment in which these issues can be examined. The AChess environment is thus first introduced, including a detailed description of the rules of the game as well as the game server at the core of the environment. We also discuss the key research issues that we expect to consider using this environment, including adversarial modeling, agent learning, multi-agent coordination, distributed problem solving and decision making, and real-time reasoning.

Given a well-defined research environment for studying real-time, adversarial reasoning, we next turn our attention to agent-based approaches playing AChess. We present the details of the agents that we have initially developed to both test the AChess environment itself as well as to provide a baseline for future approaches. These basic approaches include agents that simply select their next moves randomly, those that follow a pre-defined script of moves, and agents that use heuristics to select their course of action. We also discuss approaches we are considering for future agent development, including agents that adapt to their opponents by learning and modeling their strategies.

In order to gain a better understand how the AChess environment performs under various circumstances, a series of pilot experiments were conducted. The issues that we have focused on in these experiments involve the ability of the AChess server to respond to large numbers of movement requests, the importance of speed in decision-making by the agents, the influence of piece heterogeneity (in terms of differing piece rates) on the performance of the agents, and the robustness of the AChess server against potential cheating. The results of these experiment indicate that the AChess environment is able to support the proposed studies.

AChess provides a highly focused domain that is already widely understood, which will allow

researchers to concentrate on studying just their approach to adversarial and real-time reasoning. Such an accessible environment is essential for advancing the state-of-the-art of agent-based reasoning in real-time, adversarial domains. This is of critical importance for supporting the decision-making needs of future military systems.

2 Introduction

It is important to provide decision-makers with the ability to model and reason about activities occurring in dynamic, adversarial environments. Such activities may include predicting and evaluating enemy courses of action (eCOAs) [5, 4], competing against arbitrary opponents in adversarial gaming environments such as robotic soccer [8], protecting computer networks against ever-changing malicious attacks [9], and finding appropriate partners to cooperate with—as well as those to avoid—when forming coalitions [14, 13].

Increasingly, such activities are being examined in the context of dynamic *multi-agent systems* (MASs), where the state of the world is changing as the agents operate. In such systems, the agents must be able to adapt to their changing environment to effectively carry out their missions. An adversarial environment adds a further degree of difficulty to this process by including agents whose actions at best merely interfere with those of the core MAS, and at worst whose intentions are to thwart the goals of the system.

In order to explore the critical issues involved in reasoning and operating in a dynamic, adversarial MAS, we developed the *Asynchronous Chess* (AChess) research environment. AChess is an extension of traditional chess, with one major difference: asynchrony. It eliminates the rule that makes chess a turn-based game. With this one extension to chess, AChess becomes a rich environment to study competitive agents in real-time.

AChess is not the first real-time competitive environment, nor is it the first to examine reasoning in adversarial domains. Other efforts have considered real-time competitions in the context of simulated markets [22, 15] and robotic software [8, 20]. Similarly, there is current research in adversarial modeling to support military mission planning [5, 4] and coalition formation [14, 13]. But AChess is the first research environment that combines real-time decision making and adversarial reasoning in a problem domain that allows researchers to focus on just these core issues.

AChess is a strictly software-based agent research environment, as opposed to physical real-world environments, or any other hardware-based platforms. The most popular of the current real-time environments being used today is arguably RoboCup [8]. RoboCup is a real-time competitive environment which focuses on tackling the numerous and difficult challenges involved in enabling robots to play soccer. The challenges include real-time sensor fusion, reactive behavior, strategy acquisition, learning, real-time planning, multi-agent systems, context recognition, vision, strategic decision making, motor control, and intelligent robot control [8]. While these challenges are all very important to study, they complicate the area of research particular to real-time reasoning. Incorporating all the above listed challenges involves tremendous overhead of different subsystems for robotic competition, and takes away from the heart of the AI research that could be done.

Similarly, in the areas of military mission planning and cyberwarfare, systems such as the Emergent Adversarial Modeling System (EAMS) [5, 4] and MITRE's simulated cyberwarfare environment [9], allow users to explore various avenues for dealing with adversarial tactics. These environments support the study of automated scenario generation and dynamic update, intelligent adversarial behavior modeling, effects-based/attrition-based behavior modeling, and real-time analysis technology for comparing and grading effectiveness of alternative simulations. While they are very rich environments, they are also very complex, and thus require a researcher to possess a domain-specific military knowledge. AChess, alternatively, provides a more highly focused domain, which is already widely understood, in which researchers can concentrate on studying just their approach to adversarial and real-time reasoning.

AChess can be considered a more ideal environment to study complex decision-making because of the fact that the focus is on a smaller subset of the above mentioned areas, including **real-time reasoning**, **adversarial problems**, and **adaptive and learning approaches**. By removing the physical and sensor aspects of the RoboCup domain, as well as the complexity and domain-specific understanding needed for the military planning domains, we make the problem easier to approach, yet still rich enough to support critically important research.

This report describes the progress made to date on the development of the AChess environment, along with the research directions that this environment will support. Section 3.1 describes the AChess environment itself, including the rules, how pieces move, piece rates, and the actual simulator architecture. Section 3.2 presents the key research issues that are important to multi-agent systems research and adaptive agent solutions that exist in the AChess environment. Section 3.3 presents basic classes of agents that have been created to test the AChess environment. Section 4.1 describes a collections of experiments that we have conducted so far to examine and verify the operation of the AChess environment, while the results of those experiments are discussed in Section 4.3. Finally, in Section 5.1 we summarize the AChess project, and discuss future directions in Section 5.2.

3 Methods, Assumptions and Procedures

The first step to studying the behavior of agents and multi-agent systems in an adversarial, real-time setting is to develop a research environment in which these issues can be examined. In this section a description of the AChess environment is presented, including a discussion of the core research issues that we expect to consider using this environment. Also included is the description of a collection of simple agent-based approaches to playing AChess that have been used to test the environment, as well as to provide a baseline for future principled approaches. The results of pilot studies that have been conducted with these agents are presented in the next section.

3.1 Environment Description

AChess is very similar to standard chess, with one key difference: players do not have to wait for one another in order to make a move. From this simple change comes a rich environment in which to explore methods for time-critical reasoning. AChess is not intended to be a human playable game. Instead, the focus of this project is to develop an environment in which intelligent agents must reason not only about the current state of the chess board, but about the likelihood that their opponent will move in a specific amount of time, the cost and value of reasoning, the speed and movement of pieces, etc.

The core rules of AChess are very similar to standard chess. An AChess game uses the standard set of pieces (king, queen, rook, bishop, etc.), arranged in the conventional way at the start of the game. The key difference between chess and AChess is that players may move their pieces at any time with the following restrictions:

- Each movement must adhere to the standard chess movements for the piece. For example, rooks can only move along rows or columns.
- Pieces move at a specified rate which is defined as milliseconds per space. For example, in some configurations, it may take a pawn 800 milliseconds to move from one space to the next.
- Once a piece is in motion, its path cannot be changed.
- Pieces stop moving when they reach their destination or intersect another piece, friend or foe.

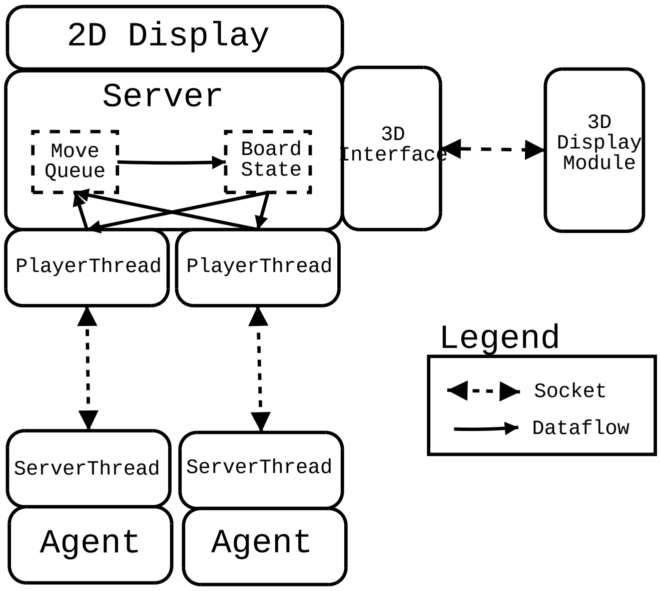


Figure 3.1: Structural diagram of the AChess simulation environment.

- If a player attempts to move a piece that is currently flagged as moving, then the piece move is rejected.

Like standard chess, pawns are promoted when they reach the opponents base row. Unlike standard chess, knights cannot "jump" their own pieces because that could lead to the condition where more than one piece occupies the same location at the same time. A final key distinction between AChess and standard chess is that play continues until one or both of the kings are killed instead of check-mated.

3.1.1 Piece Movement

The asynchrony of AChess introduces the potential for multiple pieces to move into a space at the same time. To deal with this, a special set of "deconfliction" rules were created to determine at what time a piece moves into a space, which pieces are killed, and which pieces stop when multiple pieces attempt to occupy the same board location. The deconflictions rules are as follows:

- If a player attempts to move into a space already occupied by another one of its pieces, the moving piece is stopped at its current location.
- If a player moves a piece into a space occupied by a non-moving opponent's piece, the opponents piece is killed and the players piece is stopped at that location.
- If a player attempts to move more than one piece into the same unoccupied space at the same time, one of the pieces is randomly chosen to move into the space and all others are stopped.

- If a player moves a piece into a space at the same time the opponent is attempting to move a piece into that square, one of the pieces is randomly killed and the other takes the space and is stopped.

3.1.2 Piece Rates

One of the unique characteristics of AChess is the ability to define unique movement rates for each piece type in the simulation. In doing so, a much more robust and interesting environment emerges that helps to differentiate AChess from other agent competition environments in use today. The piece rate is simply the amount of time that a piece must wait in order to move to the next square.

Once a move is submitted, given that it adheres to the above mentioned movement policies and rules, the piece is placed in its new desired location, and must then wait its $rate$ amount of time until it can move to another square. Faster rates are generally considered better as they make the pieces less vulnerable. Pieces that move at a slower rate are at a disadvantage due to the fact that they have to sit in a static position for longer than some of their enemies. Our preliminary agent experiments indicate that slow rate pieces become liabilities for offensive maneuvers, and are better suited for defensive use (see Section 4.2.3).

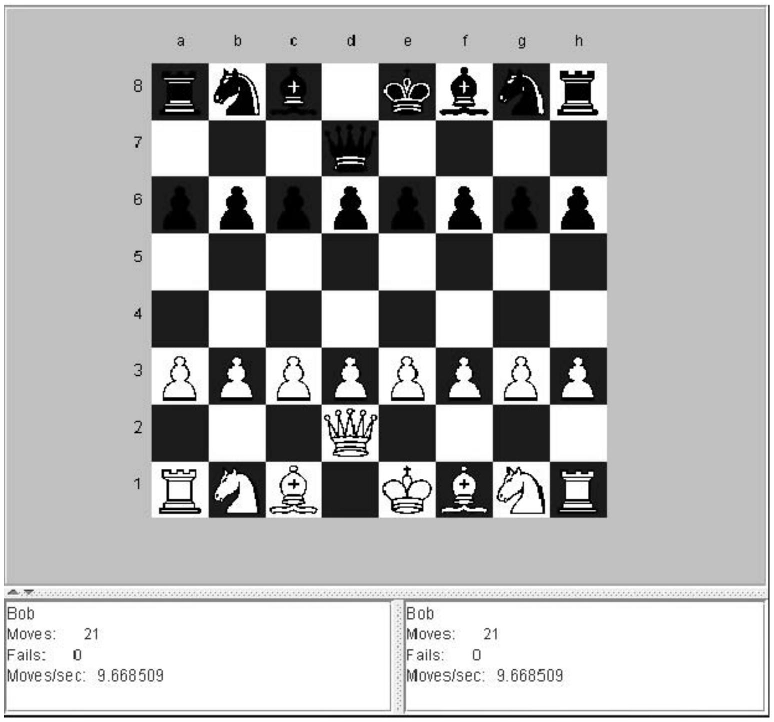
3.1.3 The Simulator

The basic AChess architecture is presented in Figure 3.1. The overall design of the system is a client-server architecture built using standard socket-based communications between the major components. There are three major pieces in the system, (1) The Server, (2) ServerThread, and (3) the 3D display module, each of which can be run on a separate computer. The entire system was implemented in Java, including the 3D display module.

3.1.3.1 The Server

The core of the system is the Server. As Figure 3.1 shows, the server is composed of a 2D display module (see Figure 3.2), two player threads for sending and receiving messages from the agents that are playing the game, and a 3D interface thread for sending state messages periodically to the 3D display module (see Figure 3.3). The server is fully parameterizable allowing the user to set the port numbers for the client connections, the size and initial layout of the board, and the specific movement rates of the pieces. In addition, the server incorporates a logging facility, which can be used to replay any game and can be used to collect metrics and performance statistics on the players.

The server is actually composed of four separate threads. The first two of these threads are for handling the socket communication with the players (annotated as PlayerThread in Figure 3.1). These player threads pull incoming "move" messages from the agents and insert them into the pending move queue. The third thread is used to push state information to a connected 3D display module.



```
Bob
Moves:     21
Fails:    0
Moves/sec:  9.668509
```

```
Bob
Moves:     21
Fails:    0
Moves/sec:  9.668509
```

Figure 3.2: The AChess 2D display.

The fourth and most most important thread in the system is the state thread. The state thread takes pending or continuing moves from the move queue and updates the current state of the game. The process by which this occurs is fairly complicated because of the asynchrony caused by the agents constantly submitting moves to the server for execution. The state thread executes at a fixed interval which can be varied as a parameter of the server. Any events that occur during the interval between successive executions of this thread are considered to have occurred at the same time. This is important because the timing of this thread can greatly impact the behavior of an AChess game by either discretizing too many or too few events during a fixed period of game time.

Certainly one of the main purposes of the state thread is to enforce the rules of the game. To do this, on each execution of this thread, the server performs the following sequence of actions:

1. The server calculates the set of pieces moving into each space during this execution of the cycle.

2. The server then checks to ensure that the move is legal (prevents pawns from capturing forward for example).

3. The server deconflicts all moves where a single player is moving multiple pieces into one space. This leaves at most one pending move into a space per player.

4. The server deconflicts all moves where a player is moving a piece into a space occupied by one of its own pieces by stopping the move. This may remove one of the pending moves. The effects of this action are chained if necessary.
5. The piece is then moved into the space, killing any non-moving enemy piece within it.
6. If two pieces belonging each to a different team try to move to the same position, the server randomly chooses one of the pieces to occupy the space. The other piece is killed.
7. If either (or both) of the kings was killed, the game is declared over.

Once the current state of the board is computed, the 2D display is updated, statistics are calculated, and the new state of the board is pushed to the two players. In this manner, the players internal state can be kept appraised of their ever changing environment.

3.1.3.2 ServerThread

The next major component of the AChess architecture is the agent interface denoted as ServerThread (see Figure 3.1). As shown in Figure 3.1, ServerThread connects to the Server via the PlayerThread, providing the agent interface for communication to the Server. The agent interface provides a standardized way of connecting and disconnecting to the server, sending and receiving moves, and receiving state information from the server. To someone designing an agent, this interface provides seamless, thread-safe access to the current state of the game, and functions for specifying moves without the worry of the particular message encodings. Several agents, written to test the AChess game, will be discussed briefly in Section 3.3.

3.1.3.3 3D Display Module

The last major component that has been designed for the AChess game is a 3D display module (see Figure 3.3). The 3D display module was created entirely using JAVA3D. Like an agent, the 3D module connects to the server via a socket to allow it to be run on a separate machine. This was done for two reasons. First, the 3D module uses quite a bit of processing power and can therefore slow down the speed of the simulator when competing for processing resources. Second, by have this module separate from the server, a user can elect to start up and shutdown the display without impacting the current simulation run.

3.2 Research Issues

In this section, we discuss some of the intriguing research issues associated with creating agents for the AChess environment, as well as why those issues are equally important to military-critical systems. Specifically we attempt to scope why AChess will be a beneficial environment for studying single-agent and multi-agent reasoning approaches in general, as well as for examining multi-agent coordination and cooperation strategies. A multitude of issues need to be addressed in Multi-Agent

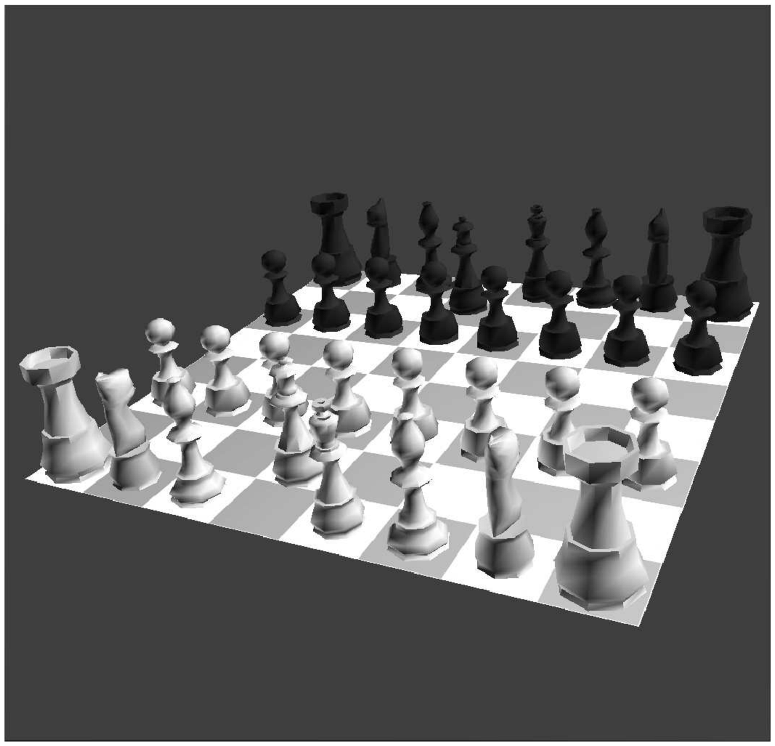


Figure 3.3: The AChess 3D display.

Systems (MASs). The same is true in the Asynchronous Chess environment when approaching viable solutions to successfully reach the goal of a defined game. Because a player connecting to the AChess server can be viewed as (or may actually be!) an MAS, we see the same challenging issues that are prevalent in MASs also appear in the AChess environment.

3.2.1 Areas of Interest

The AChess environment itself is of interest for studying the composition of an MAS. However, the real interest of this project lies in the area of the underlying intelligent agent technologies themselves. These include, but are not limited to, adversarial modeling, agent learning, multi-agent coordination, distributed problem solving and decision making, and real-time activity. We believe that these areas are of particular importance to military-critical systems, such as mission planning and distributed information systems, primarily because they address the core scientific and technical challenges for these application areas. Further, and perhaps more interestingly, we believe that lessons learned in the AChess environment will readily translate into support certain classes of physically distributed systems, such as Unmanned Autonomous Vehicles (UAVs).

As discussed in Section 2, the first of our research areas, adversarial modeling (also known as model learning [19]), is very important for providing decision-makers the ability to reason about activities occurring in dynamic environments. While adversarial modeling is of general interest in areas such as wargaming, board and card games, economic models, and others, in our case the use of adversarial modeling in the context of MASs is particularly important. Exploration of strategies for opponent modeling will play an important role in developing "new and improved" intelligent agents for AChess. The environment provides abundant support for exploring the intricacies of adversarial modeling, including modeling multi-agent interaction of the opposing team, modeling the individual agents on the team and what their reasoning process may be, and modeling potential opponent agent team plans. One way we anticipate examining adversarial modeling will be through model learning. That is, we will use reinforcement learning during game play where an agent is not only learning its own optimal behavior, but also for estimating its opponent's optimal behavior. Agents that learn are indispensable for flexibility and autonomous behavior, especially in adverse environments like war operations.

AChess also provides a rich environment for studying mechanisms for the control and coordination of teams of agents in dynamic, adversarial domains. Players in AChess may be individual agents, or they may be teams of agents collectively making decisions about the movement of pieces on the board. The coordination of many agents in a system brings forth the challenge of controlling the way in which a multi-agent team handles limited information, and percolates that information up to higher levels of understanding so that the team is successful. Success in an MAS measured in terms of completing desired goals or subgoals. In AChess, the overall goal is defined by the game. In standard AChess, capturing your opponent's king is the end goal, or end state. Coordinating a way of reaching that end goal among many agents is a continually evolving task. While many coordination algorithms exist for MASs, few address the real-time and adversarial issues that characterize AChess.

Similarly, because each AChess player may in fact be a team of agents, the environment readily supports the study of distributed decision making. Each agent may be operating on its own computing platform, requiring the team members to coordinate their activities through the available communication channels. Distributed agents may be more robust in highly dynamic environments, as they can take advantage of localized information and computational resources. However, they must also consider the cost of communication in their decision processes, the impact of reasoning with limited shared information, as well as the possibility of the opponent learning critical information from the communication.

A final research area that can be examined in the AChess environment involves agent learning. Because of the dynamic nature of AChess, agents will need to adapt their strategies to account for the behavior of their opponents. Clearly a good adversarial model will help an agent predict and adapt to its opponent's strategies, as long as they remain static. But in order for an agent to be successful against opponents that dynamically adjust their strategies over time, or against various opponents each with unique strategies, intelligent agents must be able to learn. An agent's learning mechanism must bring together the above concepts of adversarial modeling and coordination. A team that learns an optimal policy, or as close to optimal as can be found, for selecting actions in the adversarial environment is one that will easily outperform a basic agent with a pre-defined set of rules to follow. But there is a cost to learning: algorithms that learn must do so through trial and error, or exploration and exploitation. The challenge is to find ways in which agents learn fast or die.

Finally, we believe that technologies that are proven to be successful in the AChess environment will readily transition to military systems. In the rapidly changing conflicts that our military is now encountering, decision-makers need needs agents can predict an adversary's actions and provide optimal or near-optimal actions for response. Warfighters need support systems that can autonomously plan and coordinate their own activities, and even carry out missions with limited command interventions. For instance, UAVs in the sky, ground, and water, will all play important parts in our future military missions. Investigating agent technologies like the ones mentioned above, in a "soft" adversarial environment, will allow the Air Force Research Laboratory to be a leader in UAV research. Using a software environment, AChess, that poses some of the most important challenges of a real-world setting, we can safely develop new artificial intelligence algorithms and test them without putting high-cost machines or human lives at risk.

3.2.2 Research Challenges

In the previous section we discussed some general research areas that we are interested in studying in the AChess environment. These are broad areas, and a lifetime worth of work could be done in any single one of them. In this section we present some of the more immediate challenges to address not only in AChess, but MASs in general.

3.2.2.1 Metrics and Measures

Having a metric for intelligence is not only useful for comparing System A to System B, but also for comparing System A in an early stage of development to System A in a more mature stage of development [3]. This is true in the AChess environment where we are doing incremental improvements in the design of our agents. The metrics for AChess measure the intelligent decision-making ability of moves made given a bounded limit of reasoning. Theoretically, provided a quantitative performance measure exists, methods can be developed to optimize a certain system [3]. We are fortunate in that we can attempt to place a quantitative measure over the performance of the agents in AChess. A first look at AChess will tell you that there is much going on, too many variables to tweak, so we must start simple in forming metrics for measuring our agents effectiveness. Simple win/loss records could suffice, but this will give us no feedback as to *how* effective an agent did if it won, or *how* bad an agent did if it lost. If we can put an effective measure on *how* good or bad an agent did for a particular game in a match, and how that *score* will filter up to a higher level of analysis, we could provide a steady comparable mechanism for agent to agent grading. One must be careful in the assignment of an effective measure, otherwise the weakness will be emphasized throughout all the play of the game as seen in [7]. Realistically, no one set of metrics will be able to be an effective measure of an agent's worth, since games played in AChess could potentially have different assigned goals or restrictions. For instance, to loosely mimic real-world problems, taking the enemy king with minimal piece loss on both sides could be viewed as accomplishing a military task with minimal collateral damage. Measurements taken are only as useful as what can be inferred from them.

3.2.2.2 Zero-Sum to Non-Zero-Sum

Related to the above discussion of metrics, we should also introduce the issue of game classification. Our work can be viewed as a transformation from the traditional synchronized, turn-based, game of chess, to the competitive, real-time game we see in AChess. With such a transition, we are not only talking about a transition from synchronous to asynchronous, but from zero-sum to non-zero-sum games. At a coarse granularity, we can view AChess in the same zero-sum game way as its contemporary, chess, win-or-lose, 1 + (-1) = 0.

It is at a much finer granularity of observation that we transform from zero-sum game to the potentially non-zero-sum game we will want to represent in future learning agent approaches. This transformation is essential to the end-game state in that we will need a scoring system to determine not only *who* the winner was, but also quantify *how well* that particular agent did in the game.

3.2.2.3 Speed of Decision Making

An agent is just something that acts [16], but what happens when an agent not only acts, but considers what is the best action to take? It becomes rational, and acts to achieve the best outcome, or best expected outcome in times of uncertainty [16]. A clear distinction exists between *just acting* and *reasoning* about the state before acting. Successful agent implementations will carefully balance reasoning and reacting. This is important in AChess, because if an agent is not making efficient decisions, not just *rational* decisions, it may be beaten by a quicker, *less rational*, agent.

Because of the dynamic and non-deterministic (with regard to a player's opponent) nature of our environment, perceptual information overload may be too much for the processing capability of the agent. As we can see, in an adversarial environment where your opponent is making continual changes to the environment, speed of decision making is essential, but we do not want to lose quality of the decision. We can filter out parts of the perceptual information, and pay particular attention to only the parts of the information that we really need to make a quick, high-quality decision. Should we do quick sensing of the environment? Or do we only perceive our environment once we have made full sets of decisions about our previous view of what the state of the environment was? When we sense our environment, our agents should decide when to make their decisions based on the rate of change in that environment. In other words, if an agent finds that its environment is changing at a rate of X, then clearly it would be to the agent's benefit to make decisions within X - α time, α being an acceptable measure of time for submitting a decision before the environment changes again. Also, since the dynamics of the environment are not measured to be exact (i.e. server delay and such can change from update to update), we make an averaged case based reason on what X is and what α should be. The variable α should be dynamic in that it should change given how well the agents decisions were carried out in the environment, or how many of the actions that the agent submitted actually did what they were intended to do. We briefly present what we discovered about reasoning and reacting in the experimental section (4.2.2).

3.2.2.4 Coherence

Things happen in parallel in the AChess environment. Agents must develop a sense of coherence in their behaviors. Because we have a set of components acting independently of one another (i.e. both clients and the server components), coherence becomes a major issue. The agents will simultaneously perceive and act upon their environment, submitting actions to be carried out to the server. If an agent is not continuously updating its view of the environment, it may reason about incorrect information, leading to bad decisions and seemingly irrational behavior. This is a major task of the server to keep the actions that were submitted in order, and ensure that what the agent thought would be done, is done correctly. However, the mere fact that an agent made a decision given its current internal belief of what the state "looked" like is not enough to ensure deterministic results. There must be feedback so that an agent can observe the effect of an action on the environment, as well as $learn$ how quickly its opponent is making decisions and changing its environment. This can be quite different from real world models where a purely reactive agent will obtain immediate feedback from its environment. In a software system where a server is the point of main control, latency between it and its clients will create an uncertainty factor. This high amount of uncertainty and latency in the AChess environment leads to sometimes highly irrational behavior of our initial heuristic-based agent approaches.

3.2.2.5 Complexity

There is an obvious explosion of computational complexity that arises when attempting to do traditional search or game tree building. At the ply level of a search tree, we can see that the number of possible states is on the order of $n^m \cdot p^v$, where n is the total number to pieces that could possibly be moved by one team, m is the number of possible move combinations for those pieces, p is the number of pieces the adversary has on the board, and v is the number of possible moves that the adversary can move. The basic idea is that the search space is absolutely enormous and that no traditional exhaustive (or heuristic) search technique will suffice, especially with real-time constraints as in this environment. This is an open issue and probably is not a limiting factor in progress within this environment, but should be addressed.

3.3 Agent Development

While the primary focus of this report is on the AChess environment itself, an informal introduction to the agents developed to perform initial testing in the environment is necessary. These agents were developed to test the system and demonstrate logical solutions to a given game design. The current agents were developed in incremental steps, or generations, to create classes of agents. We have developed three classes of agents for AChess: *Random* agent, *Scripted* agent, and *Heuristic* agent.

After each generation of agent was designed, implemented, and debugged it was submitted for testing. Following testing it was considered final and placed in an archive to be used as a benchmark for next generation agent testing. It should be noted that each class of agent developed could be designed as either a single-agent or multi-agent implementation. Below we discuss each class of agent that was developed and the purpose for developing it in the initial environment. Empirical results addressing performance of these agents can be found in Section 4.1. Future agent classes are discussed in Section 3.3.2.

3.3.1 Existing Agent Classes

3.3.1.1 Random Agent

The **Random** agent functions by moving every available piece in a random way. Since each piece is flagged for a certain amount of time (its *rate*) as being unavailable to make another move (a *cooldown*), it is a simple determination to find the pieces to be *randomly* moved. Once those pieces are identified, their legal moves are calculated (i.e, all the moves that a piece can legally make from its current position and state) and one is chosen with equal probability. A Random agent is always moving pieces. When a game is won by the Random agent, it is a function of luck or catching the other agent off guard because it was in a "reasoning" state. This agent provides a good baseline because its strategy is unaffected by changes in the environment such as piece rate variability, which will be discussed in Section 4.2.3.

Due to the simplistic nature of the Random agent, we were able to simulate tremendous server load by submitting as many moves as possible for long periods of time (i.e. Random agents do not really try to win, so games last a longer amount of time). What we found was that overloading the server only adversely affected the player that was submitting moves too quickly. The reason why is that the thread responsible for handling the hyper-active agent's commands was effectively disabled like in a denial of service attack, while the performance of other threads on the server were not affected nearly as much.

3.3.1.2 Scripted Agent

The **Scripted** agents are given a plan—a pre-defined series of movements—which they are expected to execute. They do not deviate from the plan. All of our pre-defined strategies allowed agents to potentially win because of the speed at which a pre-defined plan can be executed and the fact that we instructed our pieces to be in typical king observed locations. Our class of scripted agents is essentially a very basic planning agent approach and does no justice to true agent planning techniques.

These agents were developed to test the stability of the system and ensure that the commands that were being submitted were being properly carried out. These agents are great tools for tutoring learning agents because of their predictable nature.

3.3.1.3 Heuristic Agent

Many different agents were developed for the class of **Heuristic** agents. In general, each agent has specialized rules for each type of pieces movement and strategy. Heuristic agents have goals and rules which tell them how they should achieve their goals based on the current state of the board.

We found these agents to be useful tools for testing the AChess environment and will be used as benchmark opponents for more advanced agent approaches.

The creation of the heuristic agents was more to find out what kinds of basic strategies would work well in an environment like AChess. Each agent in this class was developed slightly differently, and therefore had different strengths and weaknesses. Strategies developed ranged from pure brute force attempts to overpower the enemy by rushing all pieces toward the opponent's king, to lowest/greatest path cost calculations to find the opponent's king in a more elegant way. A general strategy that surfaced from all the heuristic agents developed has been to keep pieces moving whenever possible, even if there is no way to take an opponent's piece. It seems that there is an inherent reason behind this observation in that you do not want to give your opponent a steady state system to reason about. When an agent is not making moves, its opponent is not under pressure to perceive and act quickly, thus giving the adversary an advantage.

3.3.2 Principled Agent Approaches

In the previous section, we presented the approaches that have been used to date to both test the AChess environment and to determine appropriate heuristic strategies. We now turn our discussion to the topic of current agent research areas, those that we are in the process of working with.

3.3.2.1 Learning Agents

As discussed in Section 3.2.1, in order to be successful in the AChess environment, it is expected that agents will need to dynamically adapt their strategies by learning about the behaviors of their adversaries. Agent learning algorithms take many forms. Dynamic-programming, greedy strategies, Markov Decision Processes (MDP), and Q-Learning are all among popular choices. Many of these types of algorithms have been very successful in the artificial intelligence area [6]. However, most of the success has come from using these learning algorithms in a synchronized environment with tractable state spaces. AChess is an adversarial environment in which we will be applying current learning approaches to novel situations, such as exponential-sized state spaces and real-time decision making.

At a fundamental search-based level, the state space is exponential in size (see Section 3.2.2.5). Hence, representing all possible states for state valuation and action reward may not be feasible to serve the purpose of learning from experience. This also hinders the probability of seeing the same exact state twice, making a strategy like reinforcement learning (RL) insufficient in its general form. This is not to suggest that RL methods will not suffice for such environments—in fact they may do quite well—however, the way in which we approach using RL methods may need to be altered.

Iterating through the set of known states and those that have been "learned" can consume a lot of time. If spending too much time considering a state creates a situation where the state changes before an action is selected, the action will either lose value as far as effectiveness goes, or be totally invalidated. If this is not considered, then the agent can seem to act irrationally because it is making decisions which it thinks is correct, yet is not even a valid action. This is the real-time component of reasoning, and must be considered in order to make these types of learning algorithms competitive in an adversarial environment. Approaches such as Hidden Markov Models (HMM) and Markov Decision Processes (MDP) can help us approach the speed in computing action decisions quickly enough to be competitive, but these approaches have limitations inherent to their design, such as a lack of historic reference.

Other learning strategies such as GAs and NNs may prove to be extremely useful on different levels in AChess. GAs and NNs do not hold internal state space and do a traditional type search, but are trained through experience to make "rational" decisions. Case-based reasoning (CBR) is another area of interest to our research. CBR is simplistic enough to keep reasoning time to a minimal, while robust enough to handle a wide variety of generalized cases.

3.3.2.2 Hybrid Agents

As is common in highly complex domains, it is likely that no single type of reasoning will be adequate to perform well in AChess. Rather, we expect that combining important aspects of different reasoning and decision-making techniques will be needed to enable an agent to excel in such a dynamic environment. For instance, we are considering creating an AChess agent based on the work of Bowling et. al. in their work at CMU in Robot Soccer [2]. That is, if we consider an approach where reward is delayed from a series of actions that are taken, then we could utilize algorithms such as Q-learning [21]. The CMU team has taken a modified approach to building their agents, using multiple agent heuristic strategies selected via a modified Q-learning algorithm. This approach has been very competitive in the RoboCup Robot Soccer League.

Another hybrid idea that we are exploring is that of integrating multiple AI approaches together for a complete team of multi-agents. The approach would work by using certain AI methods for different piece types. For instance, there are certain heuristic strategies that work well for pawns. However, we have not found really outstanding heuristic strategies for the queen piece, so we might use RL in this case. This "hybrid-team" would be composed of heuristically scripted pawn agents, an RL based queen agent, and some other (possibly heuristic or RL based) algorithms for the other piece types.

3.3.2.3 Coordinated/Parallel MASs

As we discussed earlier in Section 3.2.1, agent solutions in AChess need not be single-agent solutions. A player in the AChess environment could be composed of multiple agents all working together as a team. The hope of an MAS AChess solution is to create a player that is greater than the sum of its parts.

One possible MAS approach to an AChess solution would be to have each agent control a specific piece. Another would be to have some agents make fast reactive decisions while other agents are able to pursue more in depth reasoning on the state of the board and adjust the global strategy of the team. There are many possibilities, but all will share the same problem of how to coordinate the parallel reasoning into coherent unified decision making.

MAS solutions with distributed decision processes may not—or in many cases, should not—perform as well as their single-agent counterparts. The coordination methods add a lot of overhead to the decision-making process. This is not a terrible thing however. The coordination problems in MAS solutions for AChess are the same coordination problems that exist in any real-world domain that involves more than one decision maker. AChess should prove to be an effective environment for developing and testing new MAS coordination technologies which could eventually be transitioned to real-world applications.

Currently we do not have an MAS solution for AChess. We are however, considering using two different MAS coordination technologies in our future agent implementations. The two technologies are the the Scalable, Periodic, Anytime Mediation protocol (SPAM) [12] and multi-agent opportunism [11].

SPAM is a protocol for agent coordination that, over time, centralizes information distributed across the agents to come up with better solutions to a problem. What makes the protocol interesting with respect to AChess is the *anytime* aspect of the protocol. This enables it to be interrupted at any point and still produce an answer. The more time it is given, the better the solution produced. But, if it must give an answer now, it can. Agent solutions using this protocol should be responsive, which is necessary in AChess, and still be capable of performing higher level decision making.

Multi-agent opportunism is a method in which individual agents pay attention to the state of the environment and attempt to identify *opportunities* in which they could assist other agents or the MAS as a whole. In AChess, a team of agents using this method could work together to accomplish each others' goals without implicit negotiation. As an example, consider the scenario where a queen is on route to take the enemy king, but there is an enemy piece in a defensive position between the queen and enemy king. If the queen is on a team using multi-agent opportunism there could be another piece on the board that recognizes this situation and is in a position to take the piece standing in the way of the queen. That piece could then take the initiative to eliminate the enemy piece allowing the queen to take the enemy king, possibly even at the expense of the goals it was itself trying to achieve.

3.4 Summary

In this section we have described the AChess research environment. We have presented the details of the server, as well as the classes of agents (both existing and planned) that can operate in this domain. We have also discussed the research topics that we believe can be readily explored in this challenging environment. In the next section, we present the details of our initial experiments in the AChess environment, and discuss how these pilot studies will assist us in our future research.

4 Results and Discussion

In the previous section we described the AChess environment in detail and discussed the many areas of research to which we believe it can contribute. In this section we investigate the actual performance of the working AChess implementation and explore some of the features and characteristics of the environment. This section begins with a description of the hardware and software used in the experimental platform, followed by a description of the global environment parameters. Finally, a description of the experiments conducted, along with their results, is provided. Note that these results are meant to analyze the AChess environment itself and not the agents that we have developed thus far.

4.1 Experimental Setup

4.1.1 Hardware and Software Configuration

For this series of experiments we ran the AChess server and the agents on three separate computer platforms connected by a gigabit local area network (see Figure 4.1). We ran one agent per computer and the AChess server on its own computer to minimize the chance of the actors interfering with each other. The three computers had identical hardware specifications, which was as follows: 3.4GHz Intel Pentium IV processor with Hyper-Threading, 2GB of system RAM, 250GB SATA150 hard drive, NIC gigabit ethernet card. All three computers were running Fedora Core 3 Linux and the agents and server were run on the Java Virtual Machine (JVM) version 1.5.0_01-b08. During the experiments the processes running on the machines were limited to the JVMs necessary for running the experiments and the necessary operating system processes. This was done to provide the agents and server with as much computing power as possible and to ensure having fair platforms for both agents.

4.1.2 Common Environment Parameters

All of the experiments we have run to date have consisted of what we call *matches*. A match is a series of 1000 games played between any two agents. Due to the random elements associated with the environment, move deconfliction and network latency, it is necessary to run multiple games to be able to draw any conclusions from the results. We chose to use 1000 games for the size of the matches because we only tested agents with static strategies and we found that 1000 was a

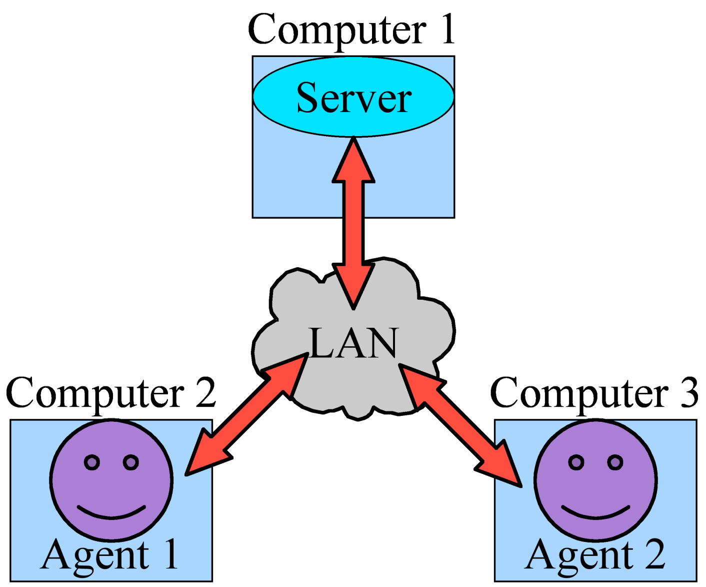


Figure 4.1: Organization of hardware configuration for all the experiments

sufficient number of games for the trends in our results to emerge.

In our experiments we used two different sets of piece rates (see Section 3.1.2). The first set, which we will call *Standard*, consists of the following rates for the chess pieces: {King 200ms, Queen 50ms, Rook 100ms, Knight 75ms, Bishop 100ms, Pawn 150ms}. *Standard* was defined arbitrarily based on what the pieces' values are in relation to one another. Higher valued pieces such as Queens and Knights were given fast rates whereas lesser valued pieces such as Rooks and Pawns were given slower rates. As the name suggests, the *Standard* set was the one used in most of our experiments (unless stated otherwise). The other set of piece rates we used in our experiments is the *Uniform* set in which all pieces have a movement rate of 100ms. *Uniform* was chosen to create an environment for the agents that is very much different from one using the *Standard* rates, providing an environment where the pieces were more balanced in their tactical utility.

For all of our experiments we set a time limit on individual games to 30 seconds. Early on in our experiments it became evident that it was necessary to have a time limit placed on the length of a game. In AChess it is not necessary for either player to move a piece. The situation where neither player determined it was appropriate to move a piece occurred quite frequently when we ran our heuristic agent against itself. This stalemate usually occurred when both kings were the last pieces left standing. The heuristics for the king are purely defensive and as such the two kings will never move to engage one another. We could have written additional heuristics to fix this situation, but we understand that this could be a problem for more than just our agents, so we included a time limit and the notion of a draw if no winner is declared by that time.

Finally, we ran all experiments with both the 2D and 3D visualizers disabled.

4.2 Experiments

To gain a better understand of how the AChess environment performs under various circumstances, we have conducted a series of pilot experiments. The issues that we have focused on in these experiments involve the ability of the AChess server to respond to large numbers of movement requests, the importance of speed in decision-making by the agents, the influence of piece heterogeneity (in terms of differing piece rates) on the performance of the agents, and the robustness of the AChess server against potential cheating.

4.2.1 Speed of Measurements

The purpose of this first series of experiments is to measure the pace at which the AChess server is able to respond to agent move requests and to determine how quickly the data the can be recorded. AChess is a simulation environment for testing real-time technologies. As such, the AChess environment must be able to respond quickly to requests and be able to record results at a fine granularity in terms of time in order to make it an effective environment for real-time testing. If AChess has the ability to record meaningful data quickly it can help ensure that agent researchers will be able explain how and why certain implementations do better than others.

Responding to move requests from agents in a timely manner is of the utmost importance. We must know the expected total amount of time it takes a move request to be received by the server, acted upon, and then finally updated in the game state of the requesting agents. The two data transmission sections in that journey are completely dependent on the latency in LAN. On our LAN we conducted a ping test by running the Linux application *ping* for two minutes between two of our computers. At the end of the test we averaged the latency reported by *ping* and found that our max and average round trip latencies were 0.272ms and 0.186ms respectively. These are acceptable numbers and can also be made stable if the three machines being used in the experiment are the only machines connected to the LAN.

The most important part of the messages journey is to be processed by the server. This step involves first being checked and accepted as a legal move for that player, put on the execution list, and then finally being acted on for the first time. To measure this we tagged all incoming moves with the current time as soon as it is received by the server. For those moves that actually made it into the execution loop of the server we measured the time difference between when the move was received by the server and the current time at the end of the move's first execution cycle. The median of the maximum times for messages to be processed was 1.358ms and the median of the average processing times was 0.120ms for a match between two of our Random agents. When combined with the total round trip times, the average response will only take about 0.306ms and that the slowest of responses should only take around 1.630ms. This shows that the AChess environment is able to respond quite fast and is suitable for testing many real-time technologies.

In this set of experiments we also wanted to show that the AChess server is able to record data quickly for real-time analysis. Data from the AChess games can be recorded not just inter-game but also intra-game. This intra-game data must be recorded with sufficient frequency during the game and should strive for minimal slow down of the server. To record data quickly and with

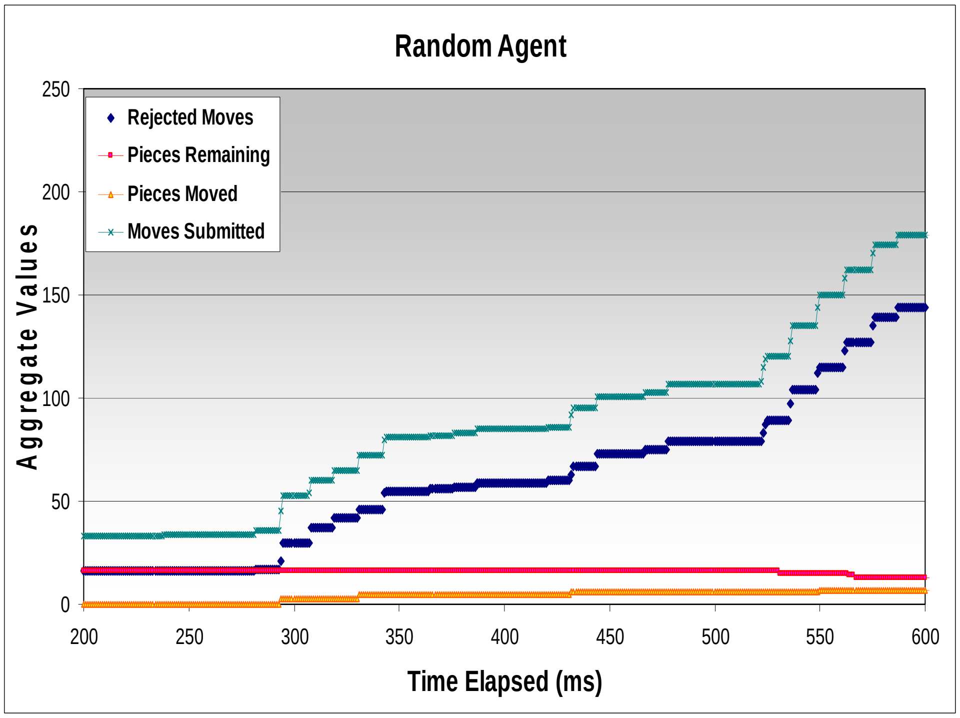


Figure 4.2: Metrics that can be gathered in the AChess environment. Shown here is a random agent's intra-game statistics.

minimal effect on the server, intra-game data is recorded at the end of every server execution cycle *to memory*, not disk. The server maintains a buffer in memory to record data in real time. The amount of memory to be used for the recording is set by the user. If the memory is exhausted during the recording of a game the server simply stops recording data for that game. The buffer is finally recorded to disk at the end of a game, before the next one starts so it does not interfere with the simulations.

Figure 4.2 is a snapshot of 400ms in a game between two Random agents. It is provided to demonstrate the degree of precision of the AChess server in collecting observations of each agent's state during the course of a game. The data in Figure 4.2 comprises the first level metrics (i.e., those that are directly measurable) that we currently record for just one of the Random agents. *Moves submitted* is simply the number of moves the agent submits to the server for execution. *Rejected moves* is the number of moves that were submitted to the server but rejected because they were not valid (i.e. went out of bounds of the board or were non-reachable for that piece type). *Pieces moved* is the total number of squares that were traversed by all pieces throughout the simulation. *Pieces remaining* is a count of what pieces are left at each time instance recorded.

It is difficult to tell from Figure 4.2, but for each metric there is a recorded value at each millisecond. This means that the AChess server was able to record data at a millisecond granularity. In reality data was actually recorded once every 0.13ms, but the AChess internal system timer only

Decision Cycle Delay	+1ms	+10ms	+50ms	+100ms	+200ms	+400ms	+800ms
Heuristic Wins	658	791	652	553	599	243	47
Random Wins	342	209	348	447	401	757	953

Table 4.1: Wins of heuristic agent against the random agent in matches where the decision cycle of the heuristic agent is artificially lengthened.

has accuracy down to a single millisecond so we were only able to report it with that frequency. The actual recording rate was obtained by dividing the number of entries recorded for a game by the total game time. Thus with a granularity of one millisecond for data recording, the AChess environment should be capable of reporting data with sufficient frequency to do perform an effective analysis for real-time research.

4.2.2 Reaction Speed

In this experiment, reaction speed in an AChess game was examined by showing that when an agent is not able to respond quickly to changes in the current game state, the agent could easily lose to an opponent that makes faster, possibly less optimal, decisions. It is absolutely imperative to the success of an agent implementation in AChess that it not only have an effective strategy but also be able to respond to unforeseen changes in the state of the board as they occur. To support our claim, a series of matches were played pitting the random agent against they heuristic agent. The decision cycle of the heuristic agent was artificially lengthened between matches, slowing its response time.

The decision cycle of the Heuristic agent was increased by introducing a sleep timer at the end of its decision loop before it could submit its moves for that cycle. This change simulates an agent taking a certain amount of time to reach a decision based on any given game state. Table 4.1 shows the decision cycle delay times for the Heuristic agent and the resulting number of wins for each agent over the matches. It is easy to see from the number of wins the Heuristic agent has that it has a better strategy than the Random agent and is able to make better decisions when the sleep time is short. However, once the timer reaches 400ms the Random agent begins to outperform the Heuristic agent even though the Heuristic agent clearly has a better strategy and makes better decisions. So, even an agent that makes really "bad" moves (Random agent) can overtake an agent that is making better decisions (Heuristic agent) if it takes too long to come to that decision. This stresses that in a real-time adversarial environment such as AChess, a decision-maker must carefully balance the amount of time it devotes to making its decision and when to actual carry out the actions. Otherwise, if the agent waits too long it will be overrun by a speedier opponent regardless of whether that opponent is making informed decisions or not. It is this observation that leads us to believe that strategies such as anytime algorithms [23] and resource-bounded reasoning [10] will be successful approaches for developing agents in AChess.

Match	Standard	Uniform
Scripted vs. Random	62.5 : 931.2	21.4 : 978.8
Heuristic vs. Random	776.1 : 224.2	594.6 : 396.3

Table 4.2: Average wins for matches between agents for the two piece movement rate sets.

4.2.3 Changing Piece Rates

For this experiment, the effect of changing piece rates on the ability of the agents to perform well in different AChess environments was studied. One of the special features of AChess is that the dynamics of the game can be changed significantly by changing the piece movement rates. It is important for technologies that emerge from AChess to work well in related domains outside of AChess. Agents developed that are able to handle different AChess environments should be easier to transition to more meaningful real-world domains, like military.

To show the effect of changing the piece rates on agents, sets of 5 matches were conducted pitting the Scripted and Heuristic agents against the Random agent using the two different sets of piece rates discussed earlier, *Standard* and *Uniform*. The results of this experiment can be seen in Table 4.2. The Scripted and Heuristic agents were both designed for the *Standard* piece rates, while the Random agent was not—it is just random. The results show that the Heuristic and Scripted agents perform best under the *Standard* piece rates and perform significantly worse under the *Uniform* piece rates. Although these results are intuitive, they still emphasize two key points: that agents with a static strategy perform poorly under different AChess settings, and that the AChess simulation can be changed by modifying the piece rates to make distinctly different environments.

In general, modifying the piece rates provides an excellent opportunity for research in adaptive agent strategies. Two such strategies being considered are from Michael Bowling's work on adaptive play selection[2] and Richard Sutton's work on prediction methods using temporal differencing [18]. Adaptive play selection involves determining methods to score and rate plays between games to improve play selection. Temporal differencing methods are a unique subset of RL algorithms that do action selection adaptation in real-time.

4.2.4 Cheating in AChess

AChess is a competitive environment and because of this we have to ask the question: Is it possible to cheat in the AChess environment? Since competing agents should not communicate with one another, and since they operate on their own respective machines, the only method for subterfuge one could reasonably expect would be for the agents to do something to the AChess server. The only action the agents in the AChess environment are capable of carrying out is to send move requests to the server. So, the question we propose is, is it possible for an agent to cheat and gain an advantage by sending a large amount of move requests to the server in an attempt of slowing the server down, giving the agent more time to analyze and act on the current state of the board?

	Heur vs. Cheat	Heur vs. Heur	Cheat vs. Cheat
Win-Win-Tie	983-13-4	489-490-21	498-490
Moves*	71, 17	49, 49	63, 50
Moves Submitted*	279, 14233	189.5, 192	51495.5, 51285

Table 4.3: Results of the cheating matches. All times are milliseconds. * = Median.

For this experiment we would like to show that an agent trying to gain an advantage by slowing down the server with meaningless move requests will not gain a competitive advantage. To test this we cloned one of our heuristic agents and made one modification: when the board state has not changed the agent, named "Cheater", bombards the server with bogus move requests instead of yielding like the standard Heuristic agent. The cheating should not interfere with the performance of the agent itself because it is being done while the agent would not be doing anything. We ran three matches for the experiment: two homogeneous matches (Cheater vs. Cheater and Heuristic vs. Heuristic) and one heterogeneous match (Heuristic vs. Cheater). From Table 4.3 we see the outcome of the Heuristic vs. Cheater match ended with a win/loss ratio 983:13, with 4 draws, in favor of the non-cheating Heuristic agent. Obviously the Cheater's strategy had backfired. The results in Table 4.4 also show that the cheating had no effect on the speed of the server. The server was still able to processes moves at the same rate.

Our explanation for this outcome is that the Cheater agent's strategy cripples the thread on the server that is dedicated to responding to the agent's requests, and not the rest of the server. We base this theory on Table 4.3 and how the Cheater was only able to get 17 moves accepted by the server compared to the 71 by the Heuristic agent when both agents have the same logic for making decisions. From this experiment we conclude that the agents cannot cheat by directly attacking the server.

4.3 Discussion

The purpose of the evaluation presented in this section was to determine whether or not AChess is a suitable environment for real-time adversarial agent testing. Not only were we concerned with the speed and performance of the server during our evaluation, but we also wanted to know if the environment is challenging, configurable, and not exploitable.

Our first sets of experiments sought to determine if the AChess server is fast enough, in terms of response time and data recording rate, to be useful as a real-time testing environment. For this

	Heur vs. Cheat	Heur vs. Heur	Cheat vs. Cheat
Cycle Time(ms)	0.12	0.14	0.13

Table 4.4: Average server cycle time during cheating matches.

experiment we first measured the length of time it took for an agent to request a move until the server actually invoked the move. We then measured the rate at which data could be recorded during the course of a game.

What we found is that, on our experimental setup, AChess is a very responsive system able to record data at a rate more than one record per millisecond. The system had a maximum and average response time of 1.630ms and 0.306ms respectively and was able to record a data entry once every 0.13ms on average. This indicates that AChess is fast enough to do testing for many technologies with real-world real-time application domains.

The experiment that focused on the speed of an agent's ability to respond to changes in the game state was designed to show that AChess is a challenging environment for real-time agent research. It showed that an agent with a clearly superior strategy, the Heuristic agent, could be beaten by an agent that made inferior but faster decisions, the Random agent, if the superior agent took too much time to reach those decisions.

The results suggest that the designers of agent solutions in AChess must be careful that their agents properly balance the amount of time they spend thinking against the time they spend responding. Agents must have some mechanism that enables them to respond quickly to changes in the board to do well in AChess. This problem of having to balance thinking and acting makes AChess a very challenging environment. Solving the challenge effectively is not without rewards either. The balancing problem is a subproblem in almost every real-time artificial intelligence application such as operating a vehicle.

In the experiment described in Section 4.2.3, we showed that the AChess environment can be changed to make it a distinctly different environment for agents by modifying the piece rates. When we changed the piece rates from *Standard* to *Uniform*, the agents that were designed for the *Standard* rates (i.e., Heuristic and Scripted) performed substantially worse against the Random agent. The Heuristic and Scripted agents have static strategies and did not adapt to the new environment.

This ability to modify the environment is important because it makes AChess a more robust testing platform. An agent solution that only does well in one instance of AChess is not very interesting or valuable. If an agent solution is able to adapt to different AChess environments and do well, it has an increased chance of doing well in a real-world application.

Finally, in our last series of experiments we set out to show that the AChess server could not be exploited by an agent with a villainous agenda. We showed that an agent trying to get more time to reason by bombarding the server with messages to slow it down was unsuccessful. Not only was it unsuccessful, it was severely detrimental to the agent performing the attack. From this experiment we were able to determine that agents cannot do anything to the server to adversely affect the performance of the game or the other agent.

In summary the AChess environment has proven itself to be a capable in all areas that we have tested. It is responsive, challenging, configurable, and immune to attempts to cheat. Therefore, AChess is a viable platform for real-time adversarial agent development and testing.

5 Conclusion

5.1 Summary

In this report, we have examined Asynchronous Chess, an in-house effort involving a new adversarial, real-time MAS research environment, along with the current and proposed solutions to the problem. As stated previously, it is important to provide decision-makers with the ability to model and reason about activities occurring in dynamic, adversarial environments. The use of these environments will be important in providing MAS solutions to areas of military use such as distributed mission planning and UAVs. Therefore, it follows that the study of MAS solutions to this area is essential in pushing the state-of-the-art.

The first step to studying the behavior of agents and multi-agent systems in an adversarial, real-time setting is to develop a research environment in which these issues could be examined. The AChess environment was introduced, including a presentation of the core research issues that we expect to consider using this environment. We also included a description of a collection of simple agent-based approaches to playing AChess that have been used to test the environment, as well as to provide a baseline for future principled approaches. The results of pilot studies conducted with initial agent approaches were also discussed, indicating that the AChess environment can indeed support the type of studies we are interested in.

Different research topics were introduced and discussed, as well as what our proposed future directions for agent development would be. We believe that in order to gain the flexibility needed for such things as enabling UAVs to successfully navigate and perform in an environment like AChess, well designed planning and learning agents are essential. Some of the new work that has come into the field of AI research will be extremely helpful for this research. It should prove interesting to adapt current technologies to perform in the AChess environment, evaluate the performance of the agents, and explore any potential improvements that might be discovered. Also, we would like to explore combining multiple strategies together into a single hybrid solution to see if we can get the best of different approaches, while minimizing the limitations they may have.

We believe the competitive aspects of AChess should generate an enthusiastic response from persons desiring to have the best agent solution for AChess. Hopefully it will become as popular and fruitful as RoboCup. AChess is freely available and we encourage interested parties to contact any of the authors for information on obtaining a current version of the AChess server and API.

5.2 Future Work

There are three major thrusts of future work that we are pursuing. Two of these thrusts are technical and involve environment enhancements and agent development. The third thrust is in gaining interest and support from other researchers in the field.

Foremost, we believe that new agent designs will be essential to good performance in the AChess environment. As discussed in Section 3.3.2, we are excited about the possibility of hybrid approaches that will involve multiple approaches combined into a single system for experimentation in AChess, as well as approaches that adapt to adversarial strategies by learning about their opponents.

Similarly, we expect that enhancing the AChess environment itself will allow us to accommodate a broader base of approaches. The first of these modifications will be to enable the server to better interact with various multi-agent platforms. The goal is to provide support for MASs like Jade [1] or AGlobe [17] to directly interact with AChess. Then, as discussed in Sections 3.2.1 and 3.3.2, there would be an actual MAS acting as a player within the AChess environment. This will help create a very open environment for MAS solutions to any research group and the environments they prefer to work with.

In addition to the enhancements to the subsystems that drive AChess, we are also exploring different scenarios in AChess beyond the standard capture-the-king game borrowed from regular chess. In this standard scenario, an agent has two goals: defend and attack. Intuitively, the primary goal of an agent should be to protect its own king, since the agent loses when its king is captured. This scenario supports exploring the validity of ideas such as the cliché *the best offense is a good defense*. But other scenarios are also possible in AChess. For example, consider the situation where we view the opponent's pawns as civilians and the rest of the pieces as the army. Is there a strategy that minimizes civilian causalities while still reaching some goal state? Similarly, in any area of conflict, one would want to minimize damage to its own resources. In AChess this translates selecting a strategy that minimizes the number of pieces lost. Finally, we may combine the two to produce a scenario where the goal is to minimize *all* casualties while still capturing the king'. This amounts to a type of "snipe the king" versus a conventional "send them to die" approach. These are just a few scenarios that AChess is capable of simulating.

Finally, an area that is very exciting for us is the opportunity to host this work as a competition in the demo track of the annual International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS). We will provide AChess as a competitive environment for agent research teams as a means of both driving one another for advanced solutions, as well as helping to establish AChess as a widely utilized platform for adversarial, real-time MAS environments. This will also provide a great opportunity for the Air Force Research Laboratory to be involved in the forefront of real-time agent technology and gain additional world-wide exposure. This type of activity is crucial for the Air Force to continue to be viewed as a leader in the area of state-of-the-art research and development, particularly in the area of artificial intelligence.

6 References

[1] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE - A FIPA-compliant agent framework. In *Proceedings of the International Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM-99)*, pages 97–108, London, April 1999.

[2] Michael Bowling, Brett Browning, and Manuela Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04)*, 2004.

[3] R. Gao and L. Tsoukalas. Performance Metrics for Intelligent Systems: An Engineering Perspective. In *Proceedings of the 2002 PerMIS Workshop*, Gaithersburg, MD, 2002. NIST Special Publication.

[4] Duane A. Gilmour, James P. Hanna, Walter A. Koziarz, William E. McKeever, , and Martin J. Walter. High-Performance Computing for Command and Control Real-Time Decision Support. *Air Force Research Laboratory Technology Horizons*, Feb 2005. http://www.afrlhorizons.com/Briefs/Feb05/IF0407.html.

[5] Duane A. Gilmour, Lee S. Krause, Lynn A. Lehman, and Eugene Santos Jr.and Qunhua Zhao. Intent Driven Adversarial Modeling. In *Proceedings of the 10th International Command And Control Research And Technology Symposium: The Future Of C2*, McLean, VA, 2005. http://www.dodccrp.org/events/2005/10th/CD/foreword.htm.

[6] Leslie Pack Kaelbling, Michael Littman, and Andrew Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, pages 237–285, May 1996.

[7] G. Kendall and G. Whitwell. An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics. In *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*, pages 995–1002, Seoul, Korea., 2001.

[8] H. Kitano, A. Minoru, Y. Kuniyoshi, I. Noda, and O. Eiichi. RoboCup: The Robot World Cup Initiative. In *Proceedings of the 1995 JSAI AI-Symposium 95: Special Session on RoboCup*, Tokyo, Japan., 1995.

[9] Gary L. Klein. The Future for Intelligent Simulation Models. *The Edge (MITRE Publication)*, Jan 2002. http://www.mitre.org/news/the_edge/january_02/klein.html.

[10] S. Koenig. Minimax real-time heuristic search. *Artificial Intelligence*, 129:165–197, 2001.

[11] James Lawton and Carmel Domshlak. On the Role of Knowledge in Multi-Agent Opportunism. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, NY, July 2004.

[12] Roger Mailler. *A Mediation-Based Approach to Cooperative, Distributed Problem Solving*. PhD thesis, University of Massachusetts, 2004.

[13] Ruggero Morselli, Jonathan Katz, and Bobby Bhattacharjee. A Game-Theoretic Framework for Analyzing Trust-Inference Protocols. In *Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems*, Harvard University, Cambridge, MA, 4-5 June 2005.

[14] Martin Rehak, Michal Pechoucek, and Jan Tozicka. Adversarial Behavior in Multi-Agent Systems. In *Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS-05)*, Budapest, Hungary, 15-17 September 2005.

[15] Juan A. Rodreguez-Aguilar, Francisco J. Martin, Pablo Noriega, Pere Garcia, , and Carles Sierra. Towards a Test-bed for Trading Agents in Electronic Auction Markets. *AIComm*, 11(1):5–19, 1998.

[16] S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice-Hall, 2003.

[17] David Sislak, Martin Rehak, and Michal Pechoucek. A-globe: Multi-Agent Platform with Advanced Simulation and Visualization Support. *Intelligent Agent Technology*, (P2415), 2002.

[18] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(9), 1998.

[19] R.S. Sutton and A.G. Barto. *Reinforcement Learning*. The MIT Press., Cambridge, MA, 1998.

[20] S. Tadokoro, H. Kitano, T. Takahashi, I. Noda, H. Matsubara, A. Shinjoh, T. Koto, I. Takeuchi, H. Takahashi, F. Matsuno, M. Hatayama, J. Nobe, and S. Shimada. The RoboCup-Rescue Project: A Robotic Approach to the Disaster Mitigation Problem. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 4089–4094, San Francisco, CA, USA, 2000.

[21] C. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambidge,England, 1989.

[22] M.P. Wellman and P.R. Wurman. A Trading Agent Competition for the Research Community. In *Proceedings of the IJCAI-99 Workshop on Agent-Mediated Electronic Trading*, Aug 1999.

[23] S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms*. PhD thesis, University of California at Berkeley, 1993.

7 Symbols, Abbreviations and Acronyms

2D	Two (2) Dimensional
3D	Three (3) Dimensional
AChess	Asynchronous Chess
AAMAS	International Joint Conference on Autonomous Agents and Multi-Agent Systems
AFRL	Air Force Research Laboratory
AI	Artificial Intelligence
API	Application Programming Interface
CBR	Case-Based Reasoning
COA	Course of Action
EAMS	Emergent Adversarial Modeling System
eCOA	Enemy Course of Action
HMM	Hidden Markov Model
GA	Genetic Algorithm
GB	Gigabyte
GHz	Gigahertz
JVM	Java Virtual Machine
LAN	Local Area Network
MAS	Multi-Agent System
MDP	Markov Decision Process
ms	Millisecond

NN	Neural Network
RoboCup	Robotic Soccer Competition
RAM	Random Access Memory
RL	Reinforcement Learning
SATA	Serial Advanced Technology Attachment
SPAM	Scalable, Periodic, Anytime Mediation protocol
UAV	Unmanned Autonomous Vehicle