Asynchronous Chess

Nathaniel Gemelli
Information Systems Research
Air Force Research Lab.
Rome, NY 13441
gemellin@rl.af.mil

Robert Wright
Information Systems Research
Air Force Research Lab.
Rome, NY 13441
wrightr@rl.af.mil

Roger Mailler *
SRI International
333 Ravenswood Dr
Menlo Park, CA 94025
mailler@ai.sri.com

ABSTRACT

We present adversarial agent work being done 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.

KEYWORDS: adversarial environments, asynchronous environments, bounded reasoning, decision cycle, rational agents, real-time reasoning

1 Introduction and Background

Asynchronous Chess (AChess) is a new environment for developing and testing real-time reasoning technologies. 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 an incredibly rich environment to study competitive agents in real-time. We propose this as a common environment for studying strictly software-based agent solutions as opposed to physical real world environments, or any other hardware-based platforms. AChess is not the first real-time competitive environment. Similar efforts have been done with simulated markets in [6] and [9], and software based environments like RoboCup-Rescue Simulation League [7].

The most popular of current common real-time environments being used today is arguably RoboCup [8]. RoboCup, or Robot Soccer, 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]. These challenges are all very important to study, but 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.

AChess can be considered a more ideal environment to study AI in game-based competitions 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**, rather than all the physical problems that hardware based environments, such as RoboCup, try to incorporate. By removing these physical (motors) and sensor (vision) aspects of the problem, we make the problem easier to approach from a strict software reasoning view. RoboCup does have a software environment, however it still involves $fuzzy$ real-world physical issues (i.e. locations, speeds, trajectories, etc.) that make the problem more difficult. AChess is differentiated from a RoboCup type environment in that we do not have problems related to uncertainty in action. When a ball is kicked in RoboCup, there is some uncertainty as to where the ball will go. For AChess, a piece is moved and it follows the specified path. AChess also introduces heterogeneous piece types which not only makes the problem more interesting from a reasoning perspective, but also increases the complexity of the problem. RoboCup does not currently consider agent heterogeneity. Every agent has the same capabilities.

RoboCup was implemented because there was some criticism against using traditional games: chess, backgammon, Yale-shooting problem, and the Monkey-Banana game [8], in that they are too abstract and do not fit well or cannot be

*Work was done while at Cornell University

easily applied to real world problems. AChess was developed for similar reasons: real world problems are too open and difficult to capture, define, and measure applicable solutions with a game like traditional chess. AChess is not as abstract a game as chess but does keep much of the simplicity of the game making it a good tool for studying reasoning. The asynchronous extension of the problem makes it more reminiscent of a real-world problem like football or strategic war games.

Soccer is seen as a common approachable "real-world" problem that is somewhere in the middle of abstract games and domain specific real-world problems. By mimicking the game of soccer, RoboCup has become hugely popular. In a similar manner, chess was chosen as the basis for AChess because it is a universal game that can be easily approached by almost anyone, even people who do not play chess well.

The rest of this paper is presented as follows. Section 2 describes the AChess environment, including the rules, how pieces move, piece rates, and the actual simulator architecture. Section 3 will present some of the research issues that are important to multi-agent systems research and adaptive agent solutions that exist in the AChess environment. Section 4 presents basic classes of agents created to test the AChess environment. Section 5 presents some early observations and measurables from the AChess environment. Section 6 will discuss future work and Section 7 will summarize.

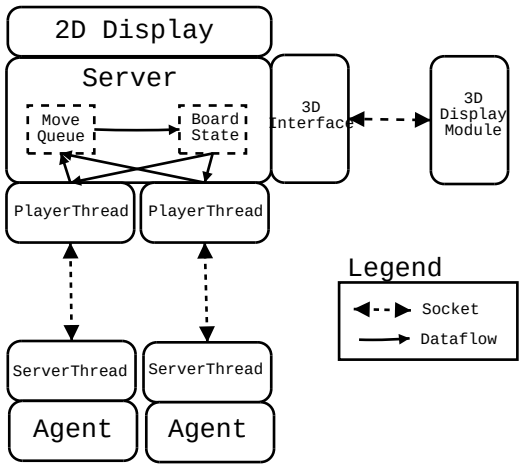

Figure 1: *A structural diagram of the AChess simulation environment.*

2 Environment Description

Asynchronous Chess 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.

2.1 The Rules

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.
- 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. Finally, play continues until one or both of the kings are killed instead of checkmated.

2.2 Piece Movement

Unlike standard chess, 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.

2.3 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. Obviously, a faster rate is a good thing as you are not a "sitting duck" so to speak. Slower move rate pieces are at a slight disadvantage due to the fact that they have to sit in a static position for longer than some of their enemies. In preliminary agent experiments, we have found that these slow rate pieces become liabilities for offensive maneuvers, and are better suited for defensive use.

2.4 The Simulator

The basic AChess architecture is presented in figure 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.

2.4.1 The Server

The core of the system is the Server. As the figure shows, the server is composed of a 2D display module (see figure 2), two player threads for sending and receiving messages

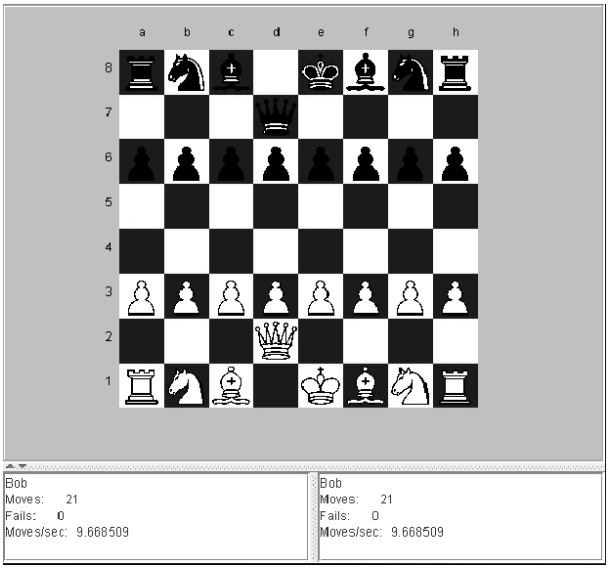

Figure 2: *The AChess 2D display.*

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). 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 the figure). 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.

The 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.

2.4.2 ServerThread

The next major component of the AChess architecture is the agent interface denoted as ServerThread (see figure 1). As shown in figure 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 4.

2.4.3 3D Display Module

The last major component that has been designed for the AChess game is a 3D display module (see figure 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 having this module separate from the server, a user can elect to start up and shutdown the display without impacting the current simulation run.

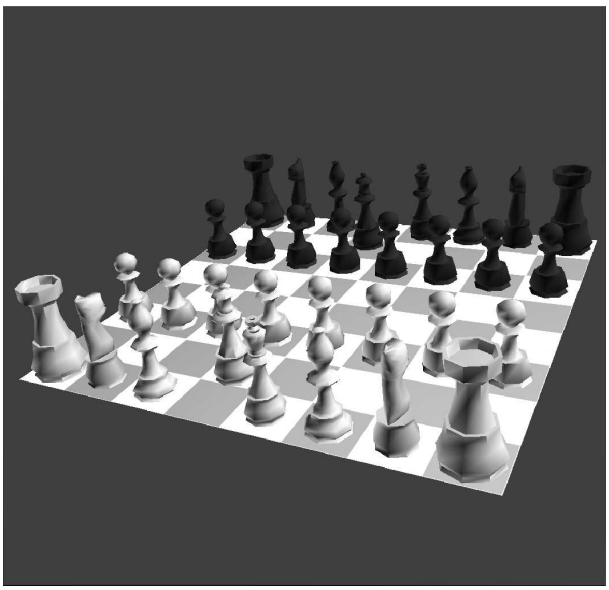

Figure 3: *A screenshot of the 3D display used for AChess.*

3 Research Issues

In this section, we will discuss some of the intriguing research issues associated with creating agents for the AChess environment. Here we attempt to scope why AChess will be a beneficial environment for studying single-agent and multi-agent reasoning approaches, as well as coordination and cooperation strategies. A multitude of issues need to be addressed in Multi-Agent Systems (MAS). 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 actually be viewed as its own MAS, we see the same challenging issues of MAS in AChess.

- **Metrics**
  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 [11]. 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 [11]. 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 [12]. 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.

- **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.

- **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.

- **Speed of Decision Making**
  An agent is just something that acts [10], 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 [10]. A clear distinction exists between $just\ acting$ and $reasoning$ about the state before acting. At what point does reasoning become futile? This is important in AChess, because if you are not making efficient decisions, not just $rational$ decisions, you may be beaten by a quicker, $less\ rational$, agent.

  Because of the dynamic and non-deterministic (with regards 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 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 what it "thinks" is a good cushion 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 (section 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.

4 Agent Development

The primary focus of this paper is on the AChess environment itself, but 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, however this is not a complete list (see Section 6):

- Random agent (rand)
- Scripted agent (script)
- Heuristic agent (heur)

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 the agent that was developed and the purpose for developing it in the initial environment development.

4.1 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, and if it happens to win a game, it is a function of luck or catching the other agent off guard because it was in a "reasoning" state. This agent provides a nice baseline because its strategy is unaffected by changes in the environment such as piece rate variability, which will be discussed in the experiment section.

$Purpose$: 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 other threads on the server were not affected nearly as much.

4.2 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.

$Purpose$: These agents were developed to test the stability of the system and ensure that the commands that were

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 1: *This table shows the wins of heuristic agent against the random agent in matches where the decision cycle of the heuristic agent is artificially lengthened. Uses SPR.*

being submitted were being properly carried out. These agents are great tools for tutoring learning agents because of their predictable nature.

4.3 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 soon to be benchmark opponents for more advanced agent approaches.

$Purpose$: 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 different, 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 opponents king, to lowest/greatest path cost calculations to find the opponents 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 opponents 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. By not making moves, you are not putting pressure on your opponent to perceive and act quickly, and giving your adversary an advantage.

5 Initial Observations

The preliminary sets of simulations that were done in the environment were to test the system and identify the immediate types of metrics that were available for collection and agent analysis. In the current AChess implementation, the API provides a certain amount of statistics about intra and inter game play. All concentrated and deep analysis on these statistics will be useful for agents doing online, adaptive learning, and/or offline, post-game analysis.

Currently available metrics for a player to gather about itself or its opponent include (but will not be limited to in future versions) intra- and inter- game simulation time, moves submitted, actual moves carried out, moves rejected, pieces remaining, pieces taken, win-loss ratios, and scoring (game defined). Below, we briefly discuss some early observations we made with our initial agents in AChess.

All simulation matches between any two agents consisted of 1000 games. We ran tests with two types of rate sets; specialized and non-specialized piece rates. The specialized piece rates (SPR) for the piece types were as follows: King 200ms , Queen 50ms, Rook 100ms, Knight 75ms, Bishop 100ms , Pawn 150ms. The non-specialized piece rates (NSPR) were set to have all piece types have a uniform rate of 100ms. For our experiments, we used one agent from each class of random, scripted and heuristic. Our heuristic and scripted agents were designed to function best for SPR. Tests were run on a single machine with following specifications: 3.4GHz Intel Pentium IV processor with Hyper-Threading, 2GB of system RAM, using Fedora Core 3.

Of particular interest is, "how the length of an agent's decision cycle determines the effectiveness of that agent in the system". In other words, can an agent that makes many poorly made decisions out perform an agent that makes fewer, but better, informed decisions in the AChess environment? To answer this question we tested our random agent against our heuristic agent and artificially lengthened the decision cycle of the heuristic agent between matches.

We increased the decision cycle of the heuristic agent by introducing a sleep timer which the agent had to wait for at the end of each iteration in its decision loop. This was done to artificially simulate a learning agents' decision cycle, as we had no readily available learning agents for AChess at the time of this writing. Table 1 shows the sleep time lengths for the heuristic agent and the resulting win/loss ratio for the matches. Even using SPR, which heuristic agents were designed to use, we can see that our heuristic agent starts to underperform when the decision cycle gets too long (approximately between the 200ms and 400ms intervals). 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, a decision maker must carefully balance the amount of time it devotes to making its decision and when to actually carry out the actions. Otherwise, if the agent waits too long it will be overrun by a speedier opponent regardless if that oppo-

Match	SPR	NSPR
script vs. rand	62.5 : 931.2	21.4 : 978.8
heur vs. rand	776.1 : 224.2	594.6 : 396.3
rand vs. rand	498.3 : 501.1	501.1 : 498.2

Table 2: *Average wins for matches between agents for the two piece movement rate sets.*

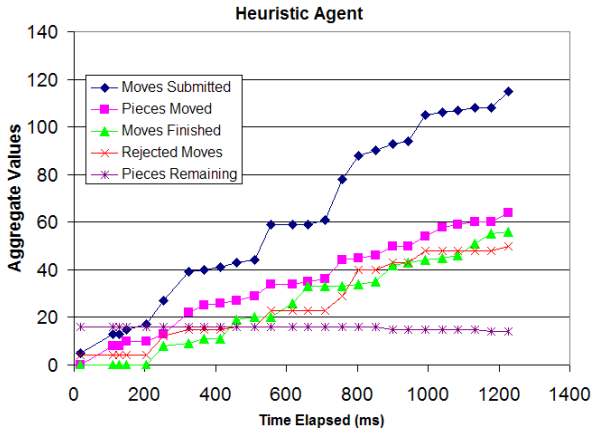

Figure 4: *Shows other metrics about which we can gather in the AChess environment. Shown here is a heuristic agents intra-game statistics.*

nent is making informed decisions or not. It is this observation that leads us to believe that strategies such as anytime algorithms [5] and resource-bounded reasoning [13] will be successful approaches for developing agents in AChess.

Figure 2 shows average win rates for the agents over multiple matches of 1000 games each match. Both the SPR and NSPR sets were tested. We can see that the agents designed to take full advantage of the SPR do not perform nearly as well in the NSPR matches as they did in matches where the SPR sets were used. Because our heuristic agents, and especially our scripted agents, were designed specifically for SPR, they are lacking a means of flexibility which will be imperative for success in AChess. In general, modifying the piece rates provides an excellent opportunity for research in adaptive agent strategies as in [3] and [4]. It also reveals that our heuristic agent is not a good solution for an environment in which conditions of the environment may change. As a side note, we see that the rand vs rand results in Figure 2 can be interpreted as showing a level of fairness in the system. Neither the white player or black player has any advantage over the other, and the rand agent is not affected by the SPR or NSPR sets.

In figure 4 we see other first level metrics that were recorded as a heuristic agent game is played. *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. *Moves finished* is the number of moves that completed without being taken or interrupted. By themselves, these statistics are not very interesting for these basic agents we have used to test the environment, but will be extremely useful for further agent development, especially in reinforcement learning techniques. An obvious extension of the current statistics will be important, and more measurables will be extracted from simulation runs as they are realized.

In conclusion, given what we have discussed in this section, we feel that AChess provides a robust platform for studying reasoning and learning algorithms in a real-time adversarial environment.

6 Future Work

The majority of the work presented in this paper has been on the agent environment of AChess itself. We are working on improvements for the AChess environment such as; support for tournament style competitions, support for and enforcement of multi-agent teams, increased speed and efficiency, and improving usability. These improvements should be available in the next version of AChess which will be released soon.

AChess as an environment is interesting in its own right, however, the future of AChess relies on the innovations that come from agent researchers using it. It is our hope that parties from all fields of artificial intelligence research will take interest in the AChess problem and environment and use it to engineer and test their own solutions. We believe the competitive aspect 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 has.

We are pursuing our own agent solutions for the AChess problem with multi-agent planning with adaptability [3] and temporal differencing methods [4]. These two technologies have proven themselves in other domains with real-time constraints and adversarial components. It should prove interesting to adapt these technologies to the AChess environment, evaluate the performance of the agents, and explore any potential improvements that might be discovered.

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.

7 Summary

We propose that AChess will become a popular common experimental platform for studying real-time competitive environments. The problem of real-time reasoning is preserved while avoiding the complicated nature of the physical world. This type of $software$ environment research is necessary in order to advance the state of real-time reasoning techniques that will eventually drive the $hardware$ implementations that exist today. In AChess, $reasoning$ is paramount.

Acknowledgment

This work has been partially funded by AFOSR and the AFRL Intelligent Information Systems Institute and under the DARPA Real program. Special thanks to Lt. Andrew Boes of the Air Force and Jeffrey Hudack for their help in developing initial test agents, and James Lawton for guidance and support.

8 References

[1] Simon, H.A. (1982). MODELS OF BOUNDED RATIONALITY (Vols. 1 and 2). Cambridge, MA: The MIT Press.
[2] D. J. Musliner, E. H. Durfee, and K. G. Shin, CIRCA: A Cooperative Intelligent Real-Time Control Architecture IEEE Transactions on Systems, Man, and Cybernetics , Vol 23 #6, 1993.
[3] 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. in press.
[4] Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. Machine Learning 3: 9-44 Machine Learning - Kluwer Academic Publishers Boston Manufactured in The Netherlands.
[5] Operational Rationality through Compilation of Anytime Algorithms S. Zilberstein. Ph.D. dissertation, Computer Science Division, University of California at Berkeley, 1993.
[6] M.P. Wellman and P.R. Wurman, "A Trading Agent Competition for the Research Community," IJCAI-99 Workshop on Agent-Mediated Electronic Trading, Aug. 1999.
[7] Tadokoro, S. and Kitano, H. and Takahashi, T. and Noda, I. and Matsubara, H. and Shinjoh, A. and Koto, T. and Takeuchi, I. and Takahashi, H. and Matsuno, F. and Hatayama, M. and Nobe, J. and Shimada, S. (2000). The RoboCup-Rescue Project: A Robotic Approach to the Disaster Mitigation Problem. Proceedings of the 2000 IEEE International Conference on Robotics and Automation, 4089-4094. San Francisco, CA, USA.
[8] Kitano, H. and Minoru, A. and Kuniyoshi, Y. and Noda, I. and Eiichi, O. (1995). RoboCup: The Robot World Cup Initiative. Proceedings of the 1995 JSAI AI-Symposium 95: Special Session on RoboCup. Tokyo, Japan.
[9] Juan A. Rodrguez-Aguilar, Francisco J. Martn, Pablo Noriega, Pere Garcia, and Carles Sierra Towards a Test-bed for Trading Agents in Electronic Auction Markets. AIComm (1998). Vol. 11;N. 1 pp. 5-19.
[10] Russell S., Norvig P., "Artificial Intelligence, A Modern Approach", Prentice-Hall, (2003), pp. 34-35.
[11] R. Gao, L. Tsoukalas, Performance Metrics for Intelligent Systems: An Engineering Perspective. Proceedings of the 2002 PerMIS Workshop. NIST Special Publication. Gaithersburg, MD.
[12] Kendall, G. and Whitwell, G. (2001). An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics. Proceedings of the 2001 IEEE Congress on Evolutionary Computation, 995-1002. Seoul, Korea.
[13] S. Koenig. Minimax Real-Time Heuristic Search. Artificial Intelligence, 129, 165-197, 2001.