Enhancing Expert System Intelligent Agents Using Game Theory

by

Craig Michael Vineyard

B.S., University of New Mexico, 2006

THESIS

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science Computer Engineering

The University of New Mexico

Albuquerque, New Mexico

December, 2008
©2008, Craig Michael Vineyard
Enhancing Expert System Intelligent Agents Using Game Theory

by

Craig Michael Vineyard

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science
Computer Engineering

The University of New Mexico
Albuquerque, New Mexico

December, 2008
Enhancing Expert System Intelligent Agents Using Game Theory

by

Craig Michael Vineyard

B.S., University of New Mexico, 2006
M.S., Computer Engineering, University of New Mexico, 2008

Abstract

An agent is generally defined as an entity capable of perceiving its environment and accomplishing a particular action without explicit instruction. Blindly taking directives without thinking is typically not intelligent, so rather a software agent is deemed intelligent if it may be characterized as situated, autonomous, flexible, and social. To behave intelligently requires decision making. Numerous fields such as economics, philosophy, and mathematics have made contributions to the realm of decision making, although none are applicable in a truly novel domain. The mathematical discipline of game theory seeks to devise the optimal strategy in strategic scenarios. Although this approach is not applicable in generalized environments it is effective in specific domains. An expert system is an intelligent agent approach to mimic human problem solving in a specific domain, and is an extension of the production system architecture. However, just as humans may take different approaches while arriving at the same solution to a problem, expert system decision making performance too is dependent upon implementation specific attempts to arrive at the
desired solution quicker. These attempts are often heuristic in nature. This thesis investigates the possibility of enhancing the underlying decision making mechanism of expert systems in a more quantitative manner by incorporating the game theoretic notion of a utility measure. A high confidence factor in taking a certain action may be solidified and validated by pairing it with a high utility as well. Not only can the addition of a utility value lead to a more preferred solution, depending upon the implementation, the utility measure may also allow an expert system to arrive at a decision quicker as courses of action with greater utility as well as confidence are considered before others. As a specific example, this thesis presents an expert system which calls offensive plays in the game of basketball depending upon the observed defense. In this example, the addition of utilities enhances the ordering of the expert system rule set, and consequently outperforms the majority of random rule orderings which is equivalent to not using utility enhancement.
Contents

List of Figures ix

1 Software Agents 1

1.1 Introduction ............................ 1
1.2 Types of Agents ........................... 2
1.3 Decision Making ............................ 3
1.4 Outline ................................ 6

2 Game Theory 7

2.1 Overview ................................ 7
2.2 Limitations ............................... 8
2.3 Mechanism Design .......................... 10

3 Production System 14

3.1 System Overview ............................ 14
3.2 Reasoning ................................ 15
## List of Figures

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Production System Architecture</td>
<td>15</td>
</tr>
<tr>
<td>4.1</td>
<td>Expert System Architecture</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>Utility Preferences</td>
<td>29</td>
</tr>
<tr>
<td>4.3</td>
<td>Utility Enhanced Decision Making</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>Random Order Decision Making</td>
<td>35</td>
</tr>
<tr>
<td>4.5</td>
<td>Utility Enhanced vs. Random Permutation in a Zone Defense</td>
<td>36</td>
</tr>
<tr>
<td>4.6</td>
<td>Utility Enhanced vs. Random Permutation when a Player is Unguarded</td>
<td>37</td>
</tr>
<tr>
<td>4.7</td>
<td>Utility Enhanced vs. Random Permutation in Man Defense</td>
<td>38</td>
</tr>
</tbody>
</table>
Chapter 1

Software Agents

1.1 Introduction

Without specifying a particular context, the term agent may be referring to anything from a spy, to a chemical reaction, to a representative. Unfortunately, even when the context is known to be that of a software agent, the ambiguity in meaning is not entirely resolved. According to Alan Kay the concept of an agent originated with John McCarthy and was later coined by Oliver Selfridge [1]. They envisioned a software entity which could simply be given a goal to carry out and it would handle the details of the necessary computer operations and could interact asking for advice if necessary [1]. Russell and Norvig later defined an agent as "...anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors [11]." Another definition according to Barbara Hayes-Roth, "Intelligent agents continuously perform three functions: perception of dynamic conditions in the environment; action to affect conditions in the environment; and reasoning to interpret perceptions, solve problems, draw inferences, and determine actions [9]." These three definitions are similar as they all generically de-
Chapter 1. Software Agents

scribe the ability of a software agent to act upon its environment without specifically
defining how to do so. However, they are also demonstrative of the fact that no
single agreed upon definition of a software agent exists. But rather, common char-
acteristics amongst the various definitions allow agents to be generalized by types,
each possessing unique capabilities.

1.2 Types of Agents

Numerous types of agents exist including, but not limited to, simple reactive agents,
agents with internal state, goal-based agents, and utility-based agents [11]. A simple
reactive agent is a collection of condition-action rules and performs predefined ac-
tions directly dependent upon the inputs from the environment. Agents with internal
state are slightly more sophisticated than simple reactive agents as they incorporate
limited memory in the form of internal states such that in addition to the basic inputs
an action is also dependent upon a stored state. Goal-based agents improve upon
agents which simply execute condition-action pairs by incorporating a desired goal to
strive for which also influences the selected action. And just as agents with internal
state improve upon simple reactive agents, utility-based agents likewise enhance the
capabilities of goal-based agents by associating a numeric value with specific states
and actions. Besides differing in their unique capabilities and functionality, not all
of these agent types truly depict the concept McCarthy originally had in mind. For
example, consider a thermostat as a simple reactive agent. A thermostat has temper-
ature sensors to perceive the environment, and heater/air conditioning control to act
upon and effect the environment. From Russell’s definition a thermostat therefore
possesses the qualities which constitute an agent with the goal of maintaining the in-
put temperature the thermostat is set at [9]. The key concept which this basic agent
is missing and separating it from the vision of McCarthy is a notion of intelligence.
Chapter 1. Software Agents

But, intelligence is a more general, harder to define concept than agents. Consequently, just as agents are often defined by agreed upon types, intelligent agents are often further characterized by specific capabilities such as being situated, autonomous, flexible, and social [5]. The concept of being situated refers to the agents ability to receive input from its environment as well as act upon and affect the environment [5]. This characteristic is in agreement with Russell’s definition by which an agent perceives its environment through sensors and may act upon the environment using effectors. Autonomy refers to the ability of an agent to act independently without requiring the involvement of another agent whether human or another intelligent agent [5]. Autonomy is therefore consistent with McCarthy’s concept of being able to handle the details of the computer operations independently. Similarly, the social trait enables McCarthy’s idea of interacting and asking for advice if necessary whether communicating with humans or other agents. And finally, the concept of flexibility may be further defined as responsive and proactive [5].

1.3 Decision Making

To be both responsive and proactive requires decision making. For example, a responsive agent responds to the environment inputs it senses. But to do so, the agent must decide how to respond as well as the most appropriate time to respond, whether it does so immediately or has time to analyze the situation. Furthermore, a proactive agent is able to take action without being specifically prompted to, if it senses an opportune scenario. Clearly this capability requires an agent be able to decide both when to take action as well as what action to take. And thus, a key design concept for multi-agent systems is the decision making mechanism. Just as humans can be indecisive and unsure of what to do in both seemingly simple situations as well as
for critical decisions, endowing software agents with the capacity to make intelligent decisions is no easier.

Furthermore, beyond simply making a decision, not all decisions are good decisions. Consequently decision making protocols are often analyzed and compared by parameters such as: negotiation time, simplicity, stability, social welfare, pareto efficiency, individual rationality, computational efficiency, and distribution and communication efficiency [4] [12]. In terms of negotiation time, it is clearly not useful for an agent to take exceedingly long periods of time to make a decision such that the decision making mechanism cannot be used in practical situations. A simple mechanism is preferred to an overly complicated architecture which in turn requires greater computational resources. An unstable design mechanism does not repeatedly arrive at the same conclusion in identical scenarios. Consequently, with unpredictable choices, an unstable design mechanism cannot be trusted to represent the user of the software agent. Social welfare is a measure of the overall value of all agents as a sum of each agent’s payoff. Pareto efficiency also views the overall global perspective in which no alternate solution benefits any individual agent. Individual rationality on the other hand pertains to each agent individually rather than collectively. For an agent to be individually rational, the resulting payoff from a decision must be no less than the the payoff an agent receives by not participating in whatever the decision at hand may be [12]. If an agent is not computationally efficient, it cannot be implemented in a realistic setting and is effectively useless. Similarly, if communication between agents and the distribution of processing between multi-agent systems is not efficient then the system will be subject to computational limitations and may not necessarily be a useful decision making mechanism.

Just as there are numerous criteria for critiquing decision making, there are likewise a wide variety of decision making theories. As stated by Herbert Simon, “...fix-
Chapter 1. Software Agents

ing agendas, setting goals, and designing actions—are usually called problem solving...evaluating and choosing, is usually called decision making [14].” One type of decision making theory is subjective expected utility (SEU), which is a mathematical technique of economics that specifies conditions for ideal utility maximization. However, SEU deals only with decision making and does not describe how to model problems, order preferences, or create new alternatives. Furthermore, SEU theory requires strong assumptions such as the consequences of all alternatives are attainable, and as a result it cannot be applied to complex real problems [14]. Rational Choice is an economic theory based upon a hypothetical ‘economic man’ who is cognizant of his environment and uses that knowledge to arrange the desired order of possible actions [13]. However, much like SEU, rational choice theory falls short as a complete decision making model because it does not specify how to perform the calculations necessary to order choices. Welfare economics analyzes the effect of resource distribution amongst members of the society as a whole [2]. The aforementioned social welfare is a measure of welfare economics which seeks to maximize the average utility of each member in the society. A similar concept is egalitarian social welfare which seeks to maximize the value of the worst member of the society. However, there are limitations which restrict the satisfiability of the members. Fundamental desirable properties of a social choice rule are: the existence of a preference ordering for all possible choices which is defined for all outcome pairs, an asymmetric and transitive ordering, a pareto efficient outcome, independence of irrelevant alternatives, and no single agent dictator dominating the preferences of others [12]. However, Arrow’s impossibility theorem asserts the impossibility of any social choice rule from satisfying all of the basic conditions just mentioned [12].
Chapter 1. Software Agents

1.4 Outline

In the remainder of this thesis, chapter two first presents a brief overview of game theory and then analyzes the applicability of game theory as a generalized decision making mechanism for intelligent software agents in novel domains. As an alternative approach to designing intelligent software, chapter three proceeds to give an overview of the production system architecture. As a modification to the production system architecture, expert systems are presented in chapter four as a particular type of intelligent software agent. The game-theoretic notion of a utility measure is then proposed as an addition to expert systems to provide a more quantitative means of enhancing expert system performance. And finally, an offensive play calling expert system for the game of basketball is provided as a particular example illustrating the effectiveness of utility enhanced expert systems.
Chapter 2

Game Theory

2.1 Overview

With so many parameters to compare decision making mechanisms, as well as the abundance of economic theories speculating the most fair allocation of limited resources, the question arises as to whether an optimal decision exists? As defined by Parsons and Woolridge game theory “...studies interactions between self-interested agents. In particular it studies the problems of how interaction strategies can be designed that will maximise the welfare of an agent in a multi-agent encounter, and how protocols or mechanisms can be designed that have certain desirable properties [8].” Game theory is also described as a collection of techniques to analyze the interaction between decision-makers using mathematics to formally represent ideas [7]. Thus, game theory serves as a technique which attempts to compute an optimal choice amongst several in a strategic interaction. And so, can game theory serve as a decision making mechanism for agents?

Although the first impression of the word “game” is often a sport, competition,
or other form of entertainment, the term game in game theory refers to a model of strategic interaction. An underlying assumption of game theory is that all decision-makers are rational. To be rational, decision-makers are assumed to be aware of all decision options, have distinct preferences, and select actions as an optimization [7]. The concept of a utility assigns numbers to the results of a particular choice. As a result, a preferential ordering may be assigned to the various choices a decision-maker may choose from. Effectively, various measures of utility relate game theory solutions to the criteria by which agent decision making mechanisms are evaluated. Dominant strategies of game theory are those actions which yield a greater utility than any other action for the particular agent regardless of the actions selected by other agents. For example, the Pareto efficiency concept is an optimal game theoretic solution by which no individual player can achieve a higher utility than by selecting their current choice. Further, other solution concepts exist within game theory such as Nash Equilibria, by which no individual player can profitably deviate from their current choice of action as long as the other players remain with their current choices [7]. The key difference between Pareto efficiency and Nash equilibria is that in Pareto optimality every player achieves their best possibly utility, whereas in Nash equilibria individual players may have alternative utilities which are higher than the current payoff, but that result is not attainable due to the choices of other players.

2.2 Limitations

However, although game theory satisfies several of the decision making design criteria previously mentioned it is subject to several limitations which prevent it from serving as the optimal general decision mechanism within intelligent agent software. While various mechanisms have been developed to handle uncertainty in the mathematical
Chapter 2. Game Theory

methods of game theory, overall it is incapable of handling an entirely unconstrained environment such as an autonomous agent would encounter where the information is sparse or full of uncertainty [15]. Bayesian games are intended to handle uncertainty regarding characteristics of opposing players [7]. For an agent to make a decision it is often necessary to reason about what opposing players will do in a given situation, and thus Bayesian games are designed to overcome the shortcoming of not knowing the type and characteristics of all other players. In this sense, uncertainty is modeled by a probability measure over a set of states [7]. Correlated equilibrium is a solution approach designed to handle uncertainty in terms of the actual state of the game. In this case, a probability space is placed over the possible states and each player has an information partition dictating what variables may be perceived. While both approaches are designed to handle uncertainty in terms of the type of opposing player and the state of the game respectively, both approaches are designed to handle scenarios in which what is unknown is known. In open situations there may be no pattern to the inconsistencies in data as well as the potential for noise in the data.

Even when the necessary data is available to perform game theoretic analysis, additional limitations also exist. Game theory assumes that it is always possible to explicitly enumerate an agent’s preferences over all possible outcomes. Depending upon the particular application, humans cannot always create consistent preference orderings necessary to formally rank all possible results [8]. Additionally, as a typical approach to counter dimensionality complexities, game theory often reduces interactions of multiple participants to interactions between two or three players by grouping collections of players, and reasoning over the conglomeration as a single player [3]. A fundamental assumption of game theory is that players are rational. This idealistic assumption does not always apply to real players, and consequently the dominant strategy predicted by game theory may not perform as predicted if the opposing players do not indeed behave entirely rationally [3]. While applicable in
Chapter 2. Game Theory

some situations, this rationality assumption is not applicable in all scenarios.

The most limiting constraint of game theory’s applicability to general multi-agent decision making is the computational efficiency evaluation criteria of decision making mechanisms. To function as a generalized decision mechanism, a game theoretic agent would have to be able to adapt to varying input requirements, opposing players, different rule sets, and unique preference relations for each game and set of players. While game theory defines various solution techniques, some of which are optimal, the solution concept varies among games and there does not exist a single solution approach applicable regardless of the game. Furthermore, game theory focuses upon the existence of solution concepts but does not specify the algorithmic techniques necessary to compute the solutions. Consequently, many game theoretic techniques assume unlimited computational resources and are often NP-Hard problems [8]. And thus, neither the computational efficiency nor the negotiation time constraints are necessarily satisfied by a game theoretic decision making mechanism.

2.3 Mechanism Design

Although game theory has not been shown to be realistic as a decision making mechanism for general-purpose intelligent agents, it can be applicable in specific scenarios where its limitations may be specifically accounted for. Within game theory, mechanism design analyzes the interaction mechanisms among rational agents with the goal of generating protocols such that desirable social outcomes are the result [12]. Consequently, the mechanism design approach does not attempt to use game theory purely as a decision making mechanism, but rather uses the results of game-theoretic analysis to constrain the strategies applied by individual players. Consequently, the intelligence behind the decision making mechanism within an agent is reduced in
Chapter 2. Game Theory

exchange for an environment which constrains the set of actions to be selected in a
game-theoretic manner.

The mechanism design approach is not applicable for all scenarios and consequently
does not allow for general-purpose agents, however auctions are an example of an
application domain which is quite suitable for mechanism design. The three different
types of auctions which differ in how the agent’s value is computed are private value,
common value, and correlated value [12]. In these auctions an agent’s value of a
good is solely dependent upon the agent’s own valuation, dependent upon the other
agent’s valuations, and partially on its own preferences as well as the other agent’s
[12]. Additionally, the different auction protocols correspond to different mecha-
nisms. An English or first-price auction allows each bidder to raise their bid until
bidding ends and the highest bidder wins. Alternatively, in a sealed-bid auction each
player submits a single bid anonymously and the highest bidder wins. A strategically
equivalent auction is known as a Dutch or descending auction by which the seller low-
ers the price until a bidder accepts the current price ending the auction. A Vickrey or
second-price sealed-bid auction requires each bidder submits a single bid, however al-
though the highest bidder wins they only pay the price of the second highest bid [12].

Mechanism design allows for dominant strategies in both the English and Vickrey
auctions. In an English auction an agent’s dominant strategy is to raise until the
agent’s value is reached [12]. In this sense, as long as an agent bids less than or
equal to its valuation of the item, then the utility received by the item is positive. If
the agent were to bid more than its valuation and win the auction, despite receiving
the item it would incur a negative net payoff. It is irrational for an agent to let
another agent win an auction with a bid lower than its valuation of the item. Rather
by simply bidding up to its personal value, the agent could win the auction and
receive a positive utility rather than the zero utility of not winning. Similarly, for
Chapter 2. Game Theory

a Vickrey auction an agent’s dominant strategy is to bid their true valuation [12]. Even though the agent pays the price of the second highest bidder rather than their own bid, if an agent were to bid more than its valuation the second highest bid may also be greater than its valuation and consequently despite winning the auction it would incur a negative payoff. As was also the case with English auctions, bidding less than the agent’s valuation may result in a payoff of zero rather than winning the bid and obtaining a positive utility. Effectively, mechanism design constrains both of these auction types such that agents should bid their actual valuations rather than speculating about how other agents assess the item or making untrue bids.

A problem prevalent in all four of the aforementioned auction protocols is collusion. Through collusion, the bidders of the auction could coordinate their bids, effectively lowering the bid necessary to win. Consequently this reduces the profit of the auctioneer. From the auctioneer’s perspective, in a Vickrey auction it can be profitable to lie if the bidders cannot verify the results. In this case by overstating the second highest bid the auctioneer profits at the expense of the winning bidder. Similarly, in open bidding auctions the auctioneer may deceivingly drive the high bid up through the use of fictitious bidders. And finally, the auctioneer may implement a reserve price by placing a bid such that the real bidders must exceed that bid [12]. Clearly mechanism design is a more feasible solution for agent decision making in applicable domains, but it too is imperfect requiring constraints to the environment.

Other non-game-theoretic approaches also exist such as general equilibrium theory, naturalistic decision making (NDM) methods, and stochastic methods. Each is applicable in some scenarios, but like game theory none serves as a general decision making mechanism. General equilibrium theory is a technique to efficiently allocate resources among multiple agents [12]. NDM attempts to take a more human approach to problem solving such as the belief-desire-intention (BDI) model based
Chapter 2. Game Theory

upon philosophical concepts of intentions, plans, and practical reasoning [6]. And finally, various stochastic approaches such as Markov decision processes and partially observable stochastic games seek to handle uncertainties in a probabilistic manner while still attempting to maximize utility.
Chapter 3

Production System

3.1 System Overview

Another attempt to attain intelligent software was in the development of the production system depicted in Figure 3.1. Utilizing a set of production rules, a memory which maintains the current state of the system, and a control structure the production system was designed by Newell and Simon to function as a computational model capable of modeling human problem solving techniques [5]. Although such a system does not satisfy all of the criteria associated with the generally accepted definition of an intelligent agent, it is a comparable concept in the sense that it was designed to reason in the same problem solving sense of a human by applying sub-actions hoping to achieve a larger goal. The set of production rules are specific to the problem domain and define what actions the system may perform [10]. The control mechanism regulates which of the set of possible actions is taken by examining the current state of memory and selecting a particular action based upon a predetermined selection algorithm. A production system is thus like an intelligent software agent as it monitors the current state of the environment stored in working
memory, and seeks to make a decision as to which action to take to solve a particular problem. The production system is an adaptive architecture and may be modified for application in varying problem domains by altering the set of production rules, however without an infinite set of rules a specific production system cannot be applied to any generalized problem. The decision making technique of most production systems is to exhaustively analyze which if any of the set of production rules may be applied towards the overall goal under the current state of the system. Although effective in particular domains, this decision selection technique may be slow and is not intelligent in discriminating between possible actions.

3.2 Reasoning

The additional capability to logically reason among applicable actions increases the general notion of intelligence in a software entity. A fundamental limitation of many game theoretic analysis techniques is the requirement of complete information regarding all possible decisions as well as opposing player’s decisions. Humans however are able to make decisions with partial information and non-exhaustive reasoning. Ab-
ductive logical inference and general abduction are alternative approaches to reasoning in uncertain situations. Logic based abduction approaches include nonmonotonic reasoning, minimum models, and set cover. The logic based approaches are often computationally infeasible. Alternatively, approaches such as the Stanford Certainty Factor Algebra, reasoning with fuzzy sets, Dempster-Shafer theory of evidence, and stochastic approaches are less computationally demanding and do not require complete information [5].

Although not as mathematically precise as formal probability theory, the Stanford Certainty Factor Algebra is generally more intuitive to human thinking. Probability theory is rigid with specific restrictions such as constraining the summation of the probability of all possible events to equal unity. The certainty factor algebra on the other hand relaxes the strict requirements and consequently is more equipped to handle imprecise human conditionals such as maybe, approximately, and most likely. The certainty factor algebra allows for a quantitative value to be assigned to intuitions without requiring a value be assigned to the converse for which there may be no intuition. Effectively, the certainty factor algebra seeks to alleviate the necessity of complete information to reason logically.
Chapter 4

Expert System

4.1 Overview

A specific application utilizing both the production system architecture and abduction is the notion of an expert system. The fundamental principle underlying an expert system is to embed domain specific knowledge regarding how to solve a particular problem within a production system such that it may reason and attempt to devise a solution with a quantifiable confidence in that decision. In this sense, by emphasizing the importance of domain knowledge as opposed to the complexity of the reasoning mechanism, an expert system exemplifies the strong method problem solving approach of artificial intelligence. Additionally, an expert system also embodies the ability to explain its reasoning by responding to ‘how’ and ‘why’ queries by the user [5]. The knowledge base and explanation subsystem illustrated in Figure 4.1 supply the extensive domain knowledge and provide the justification of the systems reasoning respectively. The inference engine performs the reasoning for the expert system just as the control structure of a production system selects among the applicable production rules.
Chapter 4. Expert System

4.2 Rule-Based, Model-Based, Case-Based

The extensive set of domain knowledge encapsulated within an expert system may be represented in a variety of ways, each with unique advantages. The most common representation approaches are rule-based, case-based, and model-based reasoning. Rule-based expert systems encode knowledge as conditional if-then pairs [5]. This representation is well suited for goal-driven inferencing by which the system begins with the problem to be solved, analyzes which rules may lead to the solution, and iterates through the knowledge-base matching problem specific facts only as necessary. Although an extensive knowledge base is required for the greatest accuracy in a rule-based system, some systems allow the user to be queried to obtain additional information necessary to solve the problem. Additionally, the rule-based knowledge representation is conducive for an explanation subsystem capable of explaining both
Chapter 4. Expert System

the line of reasoning leading to the suggested solution as well as the current reasoning. The form of rule based reasoning is not only designed to be analogous to human reasoning, but it also pairs well with Stanford Certainty Factor Algebra to provide a quantitative measure of the accuracy of the predicted solutions. In this sense, each condition action pair is assigned a measure of how likely the suggested action is given the current conditions met. The aforementioned ability of the Stanford certainty algebra to handle imprecise conditionals allows for intuition based and empirically observed confidence values to be associated with taking an action given the current state.

An attempt to associate a more scientific description of the domain problem is model based reasoning. As opposed to forming a knowledge base of a collection of rules, model based reasoning seeks to define a functional description of the problem domain. As a result of employing a more abstract functional description, model based reasoning systems are typically robust and function even in unforeseen problem settings for the particular domain. A model based system typically requires a description of each component of the system as well as a structural ordering of the relation between the individual components [5]. A drawback of the functional model approach is that it often does not allow for a meaningful explanation subsystem. Additionally, a model based system is not applicable for all problems, but typically can only be applied towards scientific domains.

An alternative approach to representing the knowledge set as either a collection of rules or through a functional description is case-based reasoning by which a collection of representative scenarios and their solutions constitute the knowledge base. This technique is analogous to law cases which have already been analyzed and set a precedence for handling a current comparable scenario. The inference system must match the current scenario to the knowledge set of cases which is most similar. Unless
an exact match may be found, an existing case must be modified to try and apply it towards the current scenario. Furthermore, a dynamic system may adapt and record the results of the system so that the case database expands and the latest results are available for potential use in future scenarios. While case based reasoning does not require the intricate detail rule based reasoning requires or a functional representation as model based reasoning does, as a tradeoff the knowledge representation is more complex requiring greater storage as well as more complex processing. Additionally, by relaxing the necessity of detailed rule sets, as is also the case with model based reasoning, the explanation capabilities of case-based reasoning are limited.

4.3 Expert System as an Intelligent Agent

Regardless of the type of expert system, in general the properties characterizing a software agent are met. Kay and McCarthy’s notion of a software agent carrying out a goal and asking for advice if necessary is fundamental to the function of an expert system. As already described, an expert system is designed to try and solve a particular problem, and furthermore does so without additional interaction unless the user must be queried to obtain information necessary to make a decision. Querying the user for further information is also comparable to perceiving the environment through sensors, and the resulting decision made by the system may be interpreted as the signal for what action should be taken by the system. In this sense, an expert system also meets the criteria described by Russell and Norvig defining a software entity.

As already mentioned, simply satisfying the criteria defining an agent does not guarantee intelligence. So to also be regarded as intelligent the question arises as to whether or not an expert system is situated, autonomous, flexible, and social? The
Chapter 4. Expert System

characteristic of being situated is satisfied by the general software agent capability of perceiving the environment in the form of askable information and acting upon the environment by the selected decision signal. The askable information may be obtained from either another agent capable of answering queries or a human user and thus an expert system is also social. Once an expert system has been programmed with the knowledge of a subject matter expert, and obtained all of the necessary information from the environment to guide its decision making process the resulting decision is obtained autonomously without further intervention. To be flexible, an expert system would need to be responsive and proactive. Although an expert system may respond to the signals it receives and requires to make informed decisions, it cannot proactively analyze the environment and decide when to take action. Effectively, although an expert system does not meet all of the generally agreed upon guidelines which describe an intelligent software agent, the majority of the criteria are met and therefore an expert system may be regarded as a rudimentary intelligent agent.

4.4 Game Theoretic Enhancement

Just because an expert system has been shown to be perceived as a software agent, and an intelligent one at that, this classification by no means implies an expert system is perfect. Rather, there is much room for improvement in the system design on the path seeking to achieve true artificial software intelligence. Regardless of the knowledge base representation, the efficiency of the system performance is dependent upon the inference engine’s search algorithm. For example, a rule-based expert system performs a goal-driven state space search and a case-based system must search for the most similar case to the current scenario. When searching among possible options there are various techniques such as a depth-first or breadth-first search.
Chapter 4. Expert System

A breadth-first search examine all of the possible options at the current state of the system. A depth-first search on the other hand traverses the state space graph along a single line of reasoning until a terminal node is met either corresponding to a successful conclusion, otherwise if the line of reasoning is unsuccessful it then backtracks to consider the next line of reasoning. A breadth first search implements goal-driven state space search, and consequently it is most efficient if the first line of reasoning traversed leads to a solution so the system does not have to backtrack and try another possible sequence of actions. Consequently, the ordering of the possible rules affects the efficiency of the system.

And so the question arises as to whether an optimal strategy exists for ordering the knowledge base of rules. One approach is to order the rules such that a rule which is either highly likely to fail or that is easy to confirm is analyzed first. In effect this approach quickly reduces the search space [5]. Alternatively, in this thesis we propose game-theoretic concepts of utility measures may be incorporated to enhance the decision making performance of expert systems. By adding a utility value in addition to confidence factors a more accurate qualitative representation may be obtained. Depending upon the implementation, the utility measure is not necessarily tied to a particular payoff, but rather is a preference ordering by which the possible actions are arranged in order of desired outcome. The particular utility values assigned to the knowledge set are implementation dependent. A high confidence in taking a certain action is solidified and validated if paired with a high utility. Additionally, even though a particular action may have a high confidence value it may be strategically more beneficial to select an alternate action with a higher utility. Similarly, options with a low confidence value combined with a low utility measure are pushed below the cutoff threshold. This approach is a tradeoff meant to guide the system to a preferred solution potentially at the expense of speed. But consequently, rather than arriving at a solution which is attained as quickly as possible
and may not even be above the cutoff threshold, by considering a utility measure with each potential decision the system is more likely to arrive at a preferred solution.

There are a variety of ways in which a utility measure may be incorporated within the expert system design. In addition to assigning a confidence factor each rule could also be given a utility value. With this approach the certainty factor algebra of the expert system shell would need to be modified to use the additional parameter. Effectively, the two values could be merged in some manner such as adding them, multiplying them, or averaging them. This approach requires a modified system shell and increases in complexity if the preference ordering of one player varies depending upon the action of another player consequently requiring a unique utility set for each possible combination of actions. Another possible implementation which requires a modification to the expert system shell but does not modify the certainty factor algebra is to use the utilities to guide the inferencing engine. Rather than simply selecting the first rule which may be applied, with each decision the rule with the greatest utility could be selected. Or alternatively, the simplest implementation which does not require any modifications to the expert system shell is to use the utility ordering to arrange the rule set in the knowledge base. The graph search guiding the decision making process is dependent upon the order of the rules, and so the rules with the highest utility are always searched first. This approach does not allow for independent rule orderings based upon the opposing players actions, but may still be implemented by generalizing the various independent orderings into a single preference ordering. For instance, one potential mechanism is to average the position of the individual actions across the set of all orderings, and sort the rules according to their average position. While arranging the rule set is a known heuristic, by additionally doing so based upon the guidance of a utility preference, the ordering is defined in a more quantitative meaningful approach than basing the order on imprecise intuitions. For example, the utility orderings may be merged from
Chapter 4. Expert System

a collection of domain expert’s rankings, which are not necessarily in agreement, and in effect an averaged ordering may be implemented rather than basing the system performance off of one expert’s personal preferences.

The addition of a utility measure is not germane to all problem domains. First of all, it is subject to the game-theoretic limitation of requiring a complete preference ordering on all actions. It may not always be possible to order all of the feasible actions in all scenarios, especially as the number of possible actions increases, in which case this enhancement technique may clearly not be implemented. The conditional if-then nature of a rule based system makes the assignment of utilities most relevant under extensive-form games in which the prior players action is known as opposed to a simultaneous move strategic game. However, that being said, depending upon the problem domain incorporating a utility may be done even when the opposing player’s selected action is unknown. For domains in which the set of players are other humans or machines, as opposed to nature, the actions taken are typically more rational and less random and are often diametrically opposed. In the case of two player games, the utility measures are often diametrically opposed and constant sum as the best action for one player often corresponds to the worst action for the opposing player.

4.5 Specific Example

As a functional example of a rule based expert system incorporating a utility measure decision making enhancement, we have created a rudimentary basketball coach system termed Coach Rule-base Expert System (RES), which calls an offensive play according to specific game situations and player capabilities. As an equivalent substitution for the environment sensors of an intelligent agent, the five ‘askable’s created
Chapter 4. Expert System

are observations that if replaced by appropriate sensors could be detected allowing the expert system to call plays autonomously. Additionally, the game of basketball serves as a beneficial example because it embodies several meaningful criteria such as diametrically opposed players, a complex decision domain, and a two level decision hierarchy by which first the defense selects a play and then the offense calls a play.

The last minute game situation is defined as when there is less than a minute left in the game and the offensive team is down by two points or less. The coach would also know the capabilities of his own players, and so if he has good jump shooters he will go for the win and attempt a three point shot. Otherwise, rather than hoping for a lucky bounce, if the coach does not have exceptional three point shooters he will instead call a play looking for a two point basket tying the game so that they may go for the win in overtime.

If the offensive team is too far behind to hope for a win or else is ahead in the game, then the simple system will fail out of the last minute game scenarios and call one of the ‘traditional plays’. Within the game of basketball, there are various offensive paradigms which a coach may call plays from. For instance, a coach may be adamant about a particular offensive scheme and call particular plays effectively placing the burden upon the defense to stop it. Alternatively, the defense is often required to be in position prior to the offense revealing its intended approach. This gives the offense an alternative approach of recognizing the defensive scheme by the opposing team and selecting a play which has a high chance of succeeding. The later is the approach implemented by my expert system in which the coach determines the defensive strategy and calls a type of play which the players currently on the court have the best chance of executing. Consequently, the ‘traditional plays’ known by the simplistic expert system are dependent upon the coach recognizing the defensive scheme the opposing team is using.
Chapter 4. Expert System

A man-to-man defense is defined as one in which a single defender is guarding each player regardless of where they move on the court, and that there are not multiple defenders guarding any single player at a given instance. Furthermore, depending upon the skills of the offensive players on the floor there are two possible plays. If there are good jump shooters on the offense, then the coach calls for the offense to set a screen opening up a jump shot for one of the players. Due to the fact that each defender is focusing on an assigned offensive player, they are most likely guarding the good offensive player tightly, so by setting a screen for a teammate, an offensive player may be freed for a high percentage jump shot. Alternatively, if the offensive team does not have good jump shooters playing at the particular point in the game but the defense is guarding them in a man-to-man defense, the coach calls for a pick and roll in which the offensive players work together to move closer to the basket for a better percentage shot.

Assigning a defensive player to a particular offensive player in a one-to-one mapping is not always the best defensive approach depending upon various criteria such as the individual defensive capabilities of players, the size distribution of the defensive players, as well as other reasons. Consequently, an alternative approach is to play a zone defense in which a player guards a particular region of the floor rather than a particular player. In this case, the coach recognizes a zone defense by the fact that the same defender is not following offensive players across the floor and there are not multiple defenders guarding a single offensive player simultaneously. Once again depending upon the individual capabilities of the offensive players there are two basic plays the coach may call. If the offensive team has good outside shooters, then it is recommended to attack the zone defense by shooting long distance jumpers that the zone is not adequately guarding against. Alternatively, the spacing of the defenders in a zone defense to cover the floor opens up the inside for a dominant
Chapter 4. Expert System

low post player to post up. In this case, if the offense does not have exception jump shooters, they should pass the ball in for a post player to get a closer higher percentage shot.

While the aforementioned plays are dependent upon subjective measures of the skill of a player, there is also the possibility that the offensive team has a player which the defense considers to be better than just good. In this case, if the offensive team has a star player the defense may double team the individual player. A double team is easily recognized by the coach, in which there is not a single defender, but rather multiple players simultaneously guarding the star player. In this case, another player on the offensive team is not being guarded, and the coach advises the team to pass the ball to the open player who is not being guarded and effectively the highest percentage shot.

The rules based upon the principles previously described are given a confidence value such that the Stanford certainty factor algebra can assign an accuracy quantification to the solution found by the system. In this application, a definitive value is assigned to the last minute game situation in which the game time and score are directly observable and not open to interpretation. The other game situations and offense plays are assigned confidence values subject to the system designer. For example, lower confidence values have been assigned to the man-to-man defense scheme than a zone defense scheme because in this simplistic expert system it generalizes all forms of zone defenses where in reality a more sophisticated system would explicitly differentiate between different types of zone defenses. Consequently, a zone defense may have the appearance of a man-to-man defense when in reality it is a complication zone defense targeting a specific player. Furthermore, the last minute game strategies have lower confidence factors than regular game scenarios as in the last minute when the game is on the line the defensive team is going to play tighter defense and
may alter their strategy.

In the current rule base, the only scenarios which yield no play call are those which do not make physical sense such as a player being guarded individually by a single defender as well as simultaneously by multiple defenders. This is a logical contradiction. If so desired, an additional rule could be added to prevent any error states and to ensure some play is called. For example the default if all else fails play could be to call time out.

The implementation of Coach RES uses the PROLOG rule base code given in Appendix A. Additionally, this particular implementation uses the PROLOG Expert System shell ExSHELL created by Dr. George Luger and William Stubblefield included in Appendix B. This expert system is designed to vary its performance in accordance to the abilities of the players it is emulating such as whether the team consists of skilled jump shooters or post players. This specific example is based upon the perspective that the offensive team has excellent jump shooters. The utility measure decision making enhancement is incorporated by strategically ordering the rules based upon average utility. As previously mentioned, this implementation does not require any modifications to the system shell.

The Utility Preference Figure 4.2 shows utility orderings assigned for each possible defense. In this domain, the utility orderings of the offense and defense are diametrically opposed and so only the preferences of the offense are listed. In the last minute game scenario the play win_on_3 is assigned the greatest utility as it seeks to win the game. The post_up and pick_roll plays are assigned the lowest utilities as these plays do not capitalize on the jump shooting abilities of the offensive team while the game is on the line. In the last minute game scenario, the free_player option, in which a wide open player shoots, is given a medium utility as the
Chapter 4. Expert System

<table>
<thead>
<tr>
<th>Offense x Defense</th>
<th>last min</th>
<th>man D</th>
<th>zone D</th>
<th>double team</th>
<th>Sum</th>
<th>Average</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>win on 3</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>20</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>tie overtime</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>14</td>
<td>3.5</td>
<td>3</td>
</tr>
<tr>
<td>screen shot</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>17</td>
<td>4.25</td>
<td>4</td>
</tr>
<tr>
<td>free player</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>25</td>
<td>6.25</td>
<td>7</td>
</tr>
<tr>
<td>shoot jumper</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>20</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>pick roll</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>9</td>
<td>2.25</td>
<td>2</td>
</tr>
<tr>
<td>post up</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>1.75</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 4.2: Utility Preferences

A guarded player is not necessarily a good shooter and may not be the best person to take the potential game winning shot. For all other defenses, the free player option is assigned the highest utility as a wide open player generally is a good offensive option. As a result, the free player action is attains the greatest average utility and is the first rule examined by the expert system. Conversely, the post_up option is assigned a utility of one for all defenses except double team as a post up game is not the most efficient offensive call for a jump shooting team. However in the case of a double team, posting up is assigned a medium utility ranking even for a jump shooting team as the over emphasis by the defense in stopping the exceptional jump shooters draws the defense away from the basket making post_up a more appealing option even for a jump shooting team. The win_on_3 and shoot_jumper offensive plays consistently were assigned above average utilities as both pertain to utilizing the shooting capability of the offensive team. Although the average utilities of these two options are equivalent, the shoot_jumper option ranks above win_on_3 option considering shoot_jumper is more versatile and may include both two or three point shot attempts whereas win_on_3 is restricted to attempting a three point shot. And thus, by sorting the rules based upon the average utility rankings, the enhanced decision making mechanism first attempts to call either free_player or shoot_jumper, while resorting to pick_roll and post_up as last resorts.
Chapter 4. Expert System

4.6 Analysis

Not only is the system dynamic based upon the players being represented, but the system performance is also dependent upon the particular defense being employed. For the sake of analysis we have arbitrarily opted to test the system performance under the assumption the opposing team is employing a zone defense. The textual output of the system including the reasoning trace leading to calling the shoot_jumper is shown in Appendix C. Furthermore, Figure 4.3 Utility Enhanced Decision Making illustrates the path through the decision space of the underlying decision process. The bold lines mark the lines of reasoning attempted moving left to right, with X’s denoting eliminated options. As shown in the given example, the free_player play is ruled out as the defense is not double teaming anyone. The second largest utility option succeeds when both the zone_D and good_shooters branches of the And-Or graph succeed. Effectively, the utility enhanced ruled ordering approach arrived at a solution in only its second line of reasoning.

Alternatively, as opposed to applying utility enhanced decision making, for comparison purposes we have randomly arranged the rule base using a pseudorandom number generator. The code implementing the resulting random ordering is included as Appendix D. Additionally the textual output resulting from running the randomly ordered rule set is shown in Appendix E. Both the utility enhanced and random ordering of the rules yield the same play call, however they traverse different paths to do so. Although the textual output depicts the same trace of reasoning to the goal, the logic paths not listed are the underlying differences in the reasoning between the enhanced and random orderings. The subtle difference in the askable user queries is a consequence of the different lines of reasoning as the random ordering requires different information than the utility enhanced ordering. Even though the textual output does not explicitly illustrate the difference in the reasoning, Figure 4.4 Ran-
Chapter 4. Expert System

dom Order Decision Making captures the additional lines of reasoning which are attempted prior to recommending the shoot jumper action. As may be seen by the figures, the random ordering of the rules arrives at the suggested action on the fifth line of reasoning examined whereas the utility enhanced ordering arrives at the same solution on the second action analyzed.

The superior performance of the utility enhanced rule ordering is not restricted to the single example of random ordering illustrated previously. We have used a random number generator to produce 100 permutations of the 7 possible actions as shown in Appendix F, where the individual rules are encoded with the given key. From this set of possible random orderings, only 16 permutations outperform the utility enhanced ordering and an addition 12 match the performance runtime. The ‘Utility Enhanced vs. Random Permutation in a Zone Defense’ Figure 4.5 illustrates the significant performance increase attained by the utility enhanced ordering as opposed to random ordering in terms of the number of possible actions considered before making a decision.

Additionally, the superior performance is not restricted to only a zone defense. We have also examined the system performance in which the defensive team is employing a man-to-man defense as well as leaving a player unguarded. As a consequence of differing defensive scheme, different plays are called by Coach RES as well. In the case of the defense double teaming a particular player and leaving another unguarded, the system recommends the open player shooting. This conclusion is the first line of reasoning examined with the utility enhanced ordering. Consequently, the best a random ordering may do is to likewise consider the free player action on the first line reasoning examined. This occurs in 15 of the random rule orderings, and thus the enhanced system outperforms the remaining 85 percent. Similarly, for a man defense, although the enhanced system requires the consideration of 4 potential
actions before setting a screen for a jump shot is called, this result still outperforms
55 percent of the random orderings. These results are seen in Figure 4.6 ‘Utility
Enhanced vs. Random Permutation when a Player is Unguarded’ and Figure 4.7
‘Utility Enhanced vs. Random Permutation in Man Defense’ figures respectively.

The utility enhanced ordering outperforms the reasoning of 72 percent of the random
rule permutations and furthermore matches 84 percent regardless of the defensive
scheme employed. Using random rule permutations removes any meaning to the
ordering, and in effect is equivalent to a system without utility enhancement. Thus,
as shown, by applying utility ordering the expert system outperforms the majority
of systems which do not employ any meaningful ordering.

In this particular application there are 5040 distinct permutations the rules may
be arranged in. Of these possible permutations, there are 720 orderings which order
the desired outcome in a single try. And thus there is a 14.29 percent chance a
random ordering outperforms the utility enhanced ordering. For a general domain
in which are there are $n$ possible actions there are $n!$ possible orderings of the set
of actions. While the number of permutation orderings increases factorially as a
function of the number of actions, conversely the likelihood of randomly selecting
the desired action as the first line of reasoning examined is a quotient of the factorial
of the number of actions. Additionally, the generalized equation representing the
likelihood that the desired action is randomly ordered as one of the first $x$ actions
may be represented as: $\frac{x!(n-x)!}{n!}$ for $n$ actions. The likelihood that the desired action
is within the first half of the possible permutations may be expressed as:

$$\frac{n}{2}!(n - \frac{n}{2})! \left(\frac{\frac{n}{2}}{n!}\right)^2$$ (4.1)
Chapter 4. **Expert System**

Taking the limit of this equation:

$$\lim_{n \to \infty} \frac{(\frac{n}{2})!^2}{n!} = 0 \quad (4.2)$$

Thus, as the number of possible actions increases the likelihood of a random ordering of the actions placing the desired action in the first half of the permutation let alone the first position goes to zero.
Utility Ordered Rules:

- offens(e(X))
  - free_player
  - shoot_jumper
    - double_team
    - zone_D good_shooter
    - not(one not(multiple_defenders)
    - not(one not(multiple_defenders)
  - tie_overtime
    - last_min not(good_shooters)
    - not(good_shooters)
    - zone_D
  - post_up
    - pick_roll
      - not(multiple_defenders)
      - not(one_defender)
    - man_D
      - not(good_shooters)
      - not(good_shooters)
  - screen_shot
    - win_on_3
      - good_shooters
      - man_D
      - less_min within_2_pts
      - less_min
      - within_2_pts
      - not(multiple_one_defender_defenders)
      - not(good_shooters)
      - not(multiple_one_defender_defenders)
Chapter 4. Expert System

Figure 4.4: Random Order Decision Making
Figure 4.5: Utility Enhanced vs. Random Permutation in a Zone Defense
Figure 4.6: Utility Enhanced vs. Random Permutation when a Player is Unguarded
Chapter 4. Expert System

Figure 4.7: Utility Enhanced vs. Random Permutation in Man Defense
Chapter 5

Conclusions

5.1 Future Work

As already mentioned, Coach RES is a relatively simplistic rule based system. However, it has potential to be expanded with a much larger rule base to incorporate much more specific offensive plays as well as the possibility of specific player profiles. By expanding the complexity of the system, factors such as memory could feasibly be incorporated if the system were run repeatedly against a particular opponent and a learning mechanism were implemented in some manner such as possibly adding additional rules specific to the opponent.

Additionally, a defensive expert system could be coupled to oppose the offensive expert system in which case analysis could be performed on a possible result of the offensive play called being run against the defensive scheme selected by the opposing system. This more sophisticated simulation could be used to further analyze the resulting decision effectiveness in addition to just the computational performance. However, the analysis would be dependent upon some sort of probabilistic measure.
Chapter 5. Conclusions

to determine the likelihood of a shot being made just because it was attempted. The main fault to such a system would be the inability to quantify the athleticism and sheer determination an exceptional player is capable of displaying.

Consequently, as a result of expanding the sophistication of Coach RES as well as developing an opposing defensive scheme caller, the resulting overall system could be embedded within a video game. As an intelligent software agent, the system would effectively be able to react to an opposing player’s actions providing the artificial intelligence of individual game characters. Additionally, not only could the system be utilized for entertainment purposes in gaming, but furthermore it could also be employed as a training tool in a simulation engine. As opposed to a video game application in which graphics and game play would be very important design characteristics, these constraints could optionally be relaxed in favor of more sophisticated decision making and play calling as a teaching tool. In this sense, given a certain scenario, coaches or player’s abilities to analyze the situation at hand and make a decision as quickly as possible could be improved outside of a real life scenario.

Although improving the sample expert system is not critical to furthering the development of the proposed utility enhanced expert system decision making mechanism, a more complex sample domain allows for more in depth analysis. As already mentioned, utility enhanced system may be implemented in a variety of ways. Modifying the expert system shell to merge the certainty factor and utility measure in a meaningful way has potential to further improve performance, however doing so would require further research into the most appropriate manner of incorporating the utility value. And additionally, various forms of utility measures may be implemented as well.
5.2 Final Decisions

Making decisions for yourself is not always an easy task. Making an intelligent agent to make decisions for you is no easier. In fact, as has been shown, even the definition of what constitutes an intelligent agent has not been universally agreed upon. Ideally, a generalized intelligent agent could be given a goal and deduce how to accomplish the goal on your behalf with minimal further instruction and interaction. But as shown, to do so would require the capability to reason and make decisions across any novel domain. Numerous fields including economics, philosophy, and the mathematics of game theory have attempted to formally analyze decision making mechanisms. Unfortunately, the results either lack computational algorithms, cannot be applied to all scenarios, are computationally intractable, or cannot be applied for other reasons.

As a result, rather than attempting to create a general purpose intelligent agent, greater success has come at the expense of generality by applying various techniques to constrained problems in specific domains. For example, although it is a mathematical method of analyzing strategic games of conflict in search of an optimal strategy, game theory has not proven particularly useful as a generalized decision mechanism. But rather, game theory may be applied in the form of mechanism design where the environment itself motivates the actions selected by individual agents. Additionally, I have incorporated game theoretic utility notions to enhance expert system decision making mechanisms. Doing so provides a more strategic means of improving expert system performance rather than relying upon inconsistent heuristic approaches. And thus, making decisions is no easy task but doing so in a more strategic rational manner has greater value than making random choices.
Appendix A

Coach RES Code

%%% Knowledge Base for a simple basketball coach expert system
%%% to call offensive plays.
%%% Craig Vineyard

% rule base:

% Top level goal, starts search.
rule((call_play(Offensive_play) :-
offense(Y), play(Y,Offensive_play)),100).

% rules to select between different offenses:

rule((offense(free_player) :-
game_situation(double_team)),95).
rule((offense(shoot_jumper) :-
game_situation(zone_D),good_shooters),75).
Appendix A. Coach RES Code

rule((offense(win_on_3) :-
game_situation(last_min),
good_shooters),70).
rule((offense(screen_shoot) :-
game_situation(man_D),good_shooters),80).
rule((offense(tie_overtime) :-
game_situation(last_min),
not(good_shooters)),60).
rule((offense(pick_roll) :-
game_situation(man_D),not(good_shooters)),80).
rule((offense(post_up) :-
game_situation(zone_D), not(good_shooters)),70).

% Rules to infer current game scenario.

rule((game_situation(last_min) :-
less_min, within_2_pts),100).
rule((game_situation(man_D) :-
one_defender, not(multiple_defenders)),70).
rule((game_situation(zone_D) :-
not(one_defender), not(multiple_defenders)),80).
rule((game_situation(double_team) :-
not(one_defender), multiple_defenders),80).

% Rules to call offensive play.

rule(play(win_on_3,
Appendix A. Coach RES Code

'Shoot a 3 pointer for the win'),100).
rule(play(tie_overtime,
    'Try for 2 points to tie the game and go to overtime'),100).
rule(play(pick_roll,
    'Run a pick and roll'),100).
rule(play(screen_shoot,
    'Set a screen to create an open jump shot'),100).
rule(play(post_up,
    'Pass the ball into the low post for a post up move'),100).
rule(play(shoot_jumper,
    'Shoot a jump shot'),100).
rule(play(free_player,
    'Pass to open player for shot'),100).

% askable descriptions

askable(less_min).
askable(within_2_pts).
askable(one_defender).
askable(multiple_defenders).
askable(good_shooters).
Appendix B

ExShell Code

%% Rule Based Expert System Shell %%%%

%%

%% This is one of the example programs from the textbook:

%%

%% Artificial Intelligence:

%% Structures and strategies for complex problem solving

%%

%% by George F. Luger and William A. Stubblefield

%%

%% Corrections by Christopher E. Davis (chris2d@cs.unm.edu)

%%

%% These programs are copyrighted by Benjamin/Cummings Publishers.

%%

%% We offer them for use, free of charge, for educational purposes only.

%%

%% Disclaimer: These programs are provided with no warranty
Appendix B. ExShell Code

%%%% whatsoever as to their correctness, reliability, or
%%%% any other property. We have written them for specific
%%%% educational purposes, and have made no effort to produce
%%%% commercial quality computer programs. Please do not expect
%%%% more of them then we have intended.
%%%%
%%%% This code has been tested with SWI-Prolog
%%%% (Multi-threaded, Version 5.2.13) and appears
%%%% to function as intended.

% solve(Goal)
% Top level call. Initializes working memory; attempts to solve
% Goal with certainty factor; prints results; asks user if they
% would like a trace.

solve(Goal) :-

   init,
   solve(Goal,C,[],1),
   nl,write('Solved '),write(Goal),
   write(' With Certainty = '),write(C),nl,nl,
   ask_for_trace(Goal).

% init
% purges all facts from working memory.

init :- retractm(fact(X)), retractm(untrue(X)).
Appendix B. ExShell Code

% solve(Goal,CF,Rulestack,Cutoff_context)
% Attempts to solve Goal by backwards chaining on rules; CF is
% certainty factor of final conclusion; Rulestack is stack of
% rules, used in why queries, Cutoff_context is either 1 or -1
% depending on whether goal is to be proved true or false
% (e.g. not Goal requires Goal be false in order to succeed).

solve(true,100,Rules,_).

solve(A,100,Rules,_) :-
    fact(A).

solve(A,-100,Rules,_) :-
    untrue(A).

solve(not(A),C,Rules,T) :-
    T2 is -1 * T,
    solve(A,C1,Rules,T2),
    C is -1 * C1.

solve((A,B),C,Rules,T) :-
    solve(A,C1,Rules,T),
    above_threshold(C1,T),
    solve(B,C2,Rules,T),
    above_threshold(C2,T),
    minimum(C1,C2,C).

solve(A,C,Rules,T) :-
Appendix B. ExShell Code

rule((A :- B), C1),
solve(B, C2, [rule(A, B, C1) | Rules], T),
C is (C1 * C2) / 100,
above_threshold(C, T).

solve(A, C, Rules, T) :-
    rule((A), C),
    above_threshold(C, T).

solve(A, C, Rules, T) :-
    askable(A),
    not(known(A)),
    ask(A, Answer),
    respond(Answer, A, C, Rules).

% respond( Answer, Query, CF, Rule_stack).
% respond will process Answer (yes, no, how, why, help).
% asserting to working memory (yes or no)
% displaying current rule from rulestack (why)
% showing proof trace of a goal (how(Goal)
% displaying help (help).
% Invalid responses are detected and the query is repeated.

respond(Bad_answer, A, C, Rules) :-
    not(member(Bad_answer, [help, yes, no, why, how(_)])),
    write('answer must be either help, (y)es, (n)o, (h)ow or (w)hy'), nl, nl,
    ask(A, Answer),

Appendix B. ExShell Code

respond(Answer,A,C,Rules).

respond(yes,A,100,_) :-
    assert(fact(A)).

respond(no,A,-100,_) :-
    assert(untrue(A)).

respond(why,A,C,[Rule|Rules]) :-
    display_rule(Rule),
    ask(A,Answer),
    respond(Answer,A,C,Rules).

respond(why,A,C,[]) :-
    write('Back to goal, no more explanation possible'),nl,nl,
    ask(A,Answer),
    respond(Answer,A,C,[]).

respond(how(Goal),A,C,Rules) :-
    respond_how(Goal),
    ask(A,Answer),
    respond(Answer,A,C,Rules).

respond(help,A,C,Rules) :-
    print_help,
    ask(A,Answer),
    respond(Answer,A,C,Rules).
Appendix B. ExShell Code

% ask(Query, Answer)
% Writes Query and reads the Answer. Abbreviations (y, n, h, w) are
% translated to appropriate command be filter_abbreviations

ask(Query,Answer) :-
    display_query(Query),
    read(A),
    filter_abbreviations(A,Answer),!.

% filter_abbreviations( Answer, Command)
% filter_abbreviations will expand Answer into Command. If
% Answer is not a known abbreviation, then Command = Answer.

filter_abbreviations(y,yes).
filter_abbreviations(n,no).
filter_abbreviations(w,why).
filter_abbreviations(h(X),how(X)).
filter_abbreviations(X,X).

% known(Goal)
% Succeeds if Goal is known to be either true or untrue.

known(Goal) :- fact(Goal).
known(Goal) :- untrue(Goal).

% ask_for_trace(Goal).
% Invoked at the end of a consultation, ask_for_trace asks the
% user if they would like a trace of the reasoning to a goal.
Appendix B. ExShell Code

ask_for_trace(Goal) :-
    write('Trace of reasoning to goal ? '),
    read(Answer),nl,
    show_trace(Answer,Goal),!.

% show_trace(Answer,Goal)
% If Answer is ‘yes’ or ‘y,’ show trace will display a trace of
% Goal, as in a ‘how’ query. Otherwise, it succeeds, doing nothing.

show_trace(yes,Goal) :- respond_how(Goal).
show_trace(y,Goal) :- respond_how(Goal).
show_trace(_,_) .

% print_help
% Prints a help screen.

print_help :-
    write('Exshell allows the following responses to queries:
         yes - query is known to be true.
         no - query is false.
        why - displays rule currently under consideration.
        how(X) - if X has been inferred,
                    displays trace of reasoning.
        help - prints this message.
    '),nl.

51
Appendix B. ExShell Code

abbreviated to first letter.'),nl,nl.

% display_query(Goal)
% Shows Goal to user in the form of a query.

display_query(Goal) :-
    write(Goal),
    write('

% display_rule(rule(Head, Premise, CF))
% prints rule in IF...THEN form.

display_rule(rule(Head, Premise, CF)) :-
    write('IF '),
    write_conjunction(Premise),
    write(' THEN '),
    write(Head),nl,
    write(' CF '),write(CF),
    nl,nl.

% write_conjunction(A)
% write_conjunction will print the components of a rule premise.
% If any are known to be true, they are so marked.

write_conjunction((A,B)) :-
    write(A), flag_if_known(A),!, nl,
    write(' AND '),
    write_conjunction(B).
Appendix B. ExShell Code

write_conjunction(A) :- write(A),flag_if_known(A),!, nl.

% flag_if_known(Goal).
% Called by write_conjunction, if Goal follows from current state
% of working memory, prints an indication, with CF.

flag_if_known(Goal) :-
    build_proof(Goal,C,_,1),
    write(' ***Known, Certainty = '),write(C).

flag_if_known(A).

% Predicates concerned with how queries.

% respond_how(Goal).
% calls build_proof to determine if goal follows from current
% state of working memory. If it does, prints a trace of
% reasoning, if not, so indicates.

respond_how(Goal) :-
    build_proof(Goal,C,Proof,1),
    interpret(Proof),nl,!.

respond_how(Goal) :-
    build_proof(Goal,C,Proof,-1),
    interpret(Proof),nl,!.
Appendix B. ExShell Code

respond_how(Goal) :-
    write('Goal does not follow at this stage of consultation.'),nl.

% build_proof(Goal, CF, Proof, Cutoff_context).
% Attempts to prove Goal, placing a trace of the proof in Proof.
% Functions the same as solve, except it does not ask for unknown
% information. Thus, it only proves goals that follow from the
% rule base and the current contents of working memory.

build_proof(true, 100, (true, 100), _).

build_proof(Goal, 100, (Goal :- given, 100), _) :- fact(Goal).

build_proof(Goal, -100, (Goal :- given, -100), _) :- untrue(Goal).

build_proof(not(Goal), C, (not(Proof), C), T) :-
    T2 is -1 * T,
    build_proof(Goal, C1, Proof, T2),
    C is -1 * C1.

build_proof((A, B), C, (ProofA, ProofB), T) :-
    build_proof(A, C1, ProofA, T),
    above_threshold(C1, T),
    build_proof(B, C2, ProofB, T),
    above_threshold(C2, T),
    minimum(C1, C2, C).

build_proof(A, C, (A :- Proof, C), T) :-
Appendix B. ExShell Code

```
rule((A :- B),C1),
build_proof(B, C2, Proof,T),
C is (C1 * C2) / 100,
above_threshold(C,T).

build_proof(A, C, (A :- true,C),T) :-
  rule((A),C),
  above_threshold(C,T).

% interpret(Proof).
% Interprets a Proof as constructed by build_proof,
% printing a trace for the user.

interpret((Proof1,Proof2)) :-
  interpret(Proof1),interpret(Proof2).

interpret((Goal :- given,C)):-
  write(Goal),
  write(' was given. CF = '), write(C),nl,nl.

interpret((not(Proof), C)) :-
  extract_body(Proof,Goal),
  write('not '),
  write(Goal),
  write(' CF = '), write(C),nl,nl,
  interpret(Proof).

interpret((Goal :- true,C)) :-
```
Appendix B. ExShell Code

write(Goal),
write(’is a fact, CF = ’),write(C),nl.

interpret(Proof) :-
    is_rule(Proof,Head,Body,Proof1,C),
nl,write(Head),write(’CF = ’),
write(C), nl,write(’was proved using the rule’),nl,nl,
rule((Head :- Body),CF),
display_rule(rule(Head, Body,CF)), nl,
interpret(Proof1).

% isrule(Proof,Goal,Body,Proof,CF)
% If Proof is of the form Goal :- Proof, extracts
% rule Body from Proof.

is_rule((Goal :- Proof,C),Goal, Body, Proof,C) :-
    not(member(Proof, [true,given])),
    extract_body(Proof,Body).

% extract_body(Proof).
% extracts the body of the top level rule from Proof.

extract_body((not(Proof), C), (not(Body))) :-
    extract_body(Proof,Body).

extract_body((Proof1,Proof2),(Body1,Body2)) :-
},extract_body(Proof1,Body1),
extract_body(Proof2,Body2).
Appendix B. ExShell Code

extract_body((Goal :- Proof,C),Goal).

% Utility Predicates.

retractm(X) :- retract(X), fail.
retractm(X) :- retract((X:-Y)), fail.
retractm(X).

member(X,[X|_]).
member(X,[_|T]) :- member(X,T).

minimum(X,Y,X) :- X =< Y.
minimum(X,Y,Y) :- Y < X.

above_threshold(X,1) :- X >= 20.
above_threshold(X,-1) :- X =< -20.
Appendix C

Utility Enhanced Output

1 ?- solve(call_play(X)).
one_defender? n.
multiple_defenders? n.
good_shooters? y.

Solved call_play(Shoot a jump shot) With Certainty = 60

Trace of reasoning to goal ? y.

call_play(Shoot a jump shot) CF = 60
was proved using the rule

IF offense(shoot_jumper) ***Known, Certainty = 60
AND play(shoot_jumper, Shoot a jump shot) ***Known, Certainty = 100
THEN call_play(Shoot a jump shot)
CF 100
Appendix C. Utility Enhanced Output

offense(shoot_jumper) CF = 60
was proved using the rule

IF game_situation(zone_D) ***Known, Certainty = 80
AND good_shooters ***Known, Certainty = 100
THEN offense(shoot_jumper)
CF 75

game_situation(zone_D) CF = 80
was proved using the rule

IF not(one_defender) ***Known, Certainty = 100
AND not(multiple_defenders) ***Known, Certainty = 100
THEN game_situation(zone_D)
CF 80

not one_defender CF = 100

one_defender was given. CF = -100

not multiple_defenders CF = 100

multiple_defenders was given. CF = -100
Appendix C. Utility Enhanced Output

good shooters was given. CF = 100

play(shoot_jumper, Shoot a jump shot) is a fact, CF = 100

X = 'Shoot a jump shot'
Appendix D

Randomly Ordered Rule Set

%%% Knowledge Base for a simple basketball coach expert system
%%% to call offensive plays.
%%% Craig Vineyard
%%% Random Rule Ordering
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% rule base:

% Top level goal, starts search.
rule((call_play(Offensive_play) :-
    offense(Y), play(Y,Offensive_play)),100).

% rules to select between different offenses:
rule((offense(pick_roll) :-
    game_situation(man_D),not(good_shooters)),80).
rule((offense(free_player) :-
    game_situation(double_team)),95).
Appendix D. Randomly Ordered Rule Set

rule((offense(screen_shoot) :-
game_situation(man_D),good_shooters),80).
rule((offense(tie_overtime) :-
game_situation(last_min),
not(good_shooters)),60).
rule((offense(shoot_jumper) :-
game_situation(zone_D),good_shooters),75).
rule((offense(win_on_3) :-
game_situation(last_min),
good_shooters),70).
rule((offense(post_up) :-
game_situation(zone_D), not(good_shooters)),70).

% Rules to infer current game scenario.

rule((game_situation(last_min) :-
less_min, within_2_pts),100).
rule((game_situation(man_D) :-
one_defender, not(multiple_defenders)),70).
rule((game_situation(zone_D) :-
not(one_defender), not(multiple_defenders)),80).
rule((game_situation(double_team) :-
not(one_defender), multiple_defenders),80).

% Rules to call offensive play.
Appendix D. Randomly Ordered Rule Set

rule(play(win_on_3,
     'Shoot a 3 pointer for the win'),100).
rule(play(tie_overtime,
     'Try for 2 points to tie the game and go to overtime'),100).
rule(play(pick_roll,
     'Run a pick and roll'),100).
rule(play(screen_shoot,
     'Set a screen to create an open jump shot'),100).
rule(play(post_up,
     'Pass the ball into the low post for a post up move'),100).
rule(play(shoot_jumper,
     'Shoot a jump shot'),100).
rule(play(free_player,
     'Pass to open player for shot'),100).

% askable descriptions

askable(less_min).
askable(within_2_pts).
askable(one_defender).
askable(multiple_defenders).
askable(good_shooters).
Appendix E

Randomly Ordered Rule Output

one_defender? n.
multiple_defenders? n.
less_min? n.
good_shooters? y.

Solved call_play(Shoot a jump shot) With Certainty = 60

Trace of reasoning to goal ? y.

call_play(Shoot a jump shot) CF = 60

was proved using the rule

IF offense(shoot_jumper) ***Known, Certainty = 60
AND play(shoot_jumper, Shoot a jump shot) ***Known, Certainty = 100
THEN call_play(Shoot a jump shot)
CF 100
Appendix E. Randomly Ordered Rule Output

offense(shoot_jumper) CF = 60
was proved using the rule

IF game_situation(zone_D) ***Known, Certainty = 80
AND good_shooters ***Known, Certainty = 100
THEN offense(shoot_jumper)
CF 75

game_situation(zone_D) CF = 80
was proved using the rule

IF not(one_defender) ***Known, Certainty = 100
AND not(multiple_defenders) ***Known, Certainty = 100
THEN game_situation(zone_D)
CF 80

not one_defender CF = 100

one_defender was given. CF = -100

not multiple_defenders CF = 100

multiple_defenders was given. CF = -100
Appendix E. Randomly Ordered Rule Output

good_shooters was given. CF = 100

play(shoot_jumper, Shoot a jump shot) is a fact, CF = 100

X = 'Shoot a jump shot'
Appendix F

Random Rule Permutations

Key:
1 = post_up
2 = pick_roll
3 = tie_overtime
4 = screen_shot
5 = win_on_3
6 = shoot_jumper
7 = free_player

100 Random Rule Orderings:

2 7 4 3 6 5 1
1 3 4 7 5 2 6
7 1 2 5 6 4 3
3 4 6 5 1 7 2
3 2 1 5 7 4 6
7 4 1 3 5 6 2
7 2 6 4 1 5 3
Appendix F. Random Rule Permutations

6 1 2 4 5 3 7
1 5 4 7 3 2 6
5 6 4 7 1 3 2
2 5 7 4 6 1 3
1 6 3 2 4 7 5
7 1 4 3 6 5 2
1 3 6 4 7 5 2
1 6 4 7 5 3 2
6 1 7 4 2 5 3
6 3 4 7 2 1 5
5 1 3 2 6 4 7
5 7 3 4 6 2 1
5 3 6 2 4 7 1
5 6 2 7 4 1 3
6 5 7 2 4 3 1
1 5 4 2 7 6 3
1 2 5 3 6 4 7
3 7 6 5 4 2 1
6 7 4 5 1 2 3
6 4 3 1 7 5 2
4 5 7 2 3 6 1
3 2 7 6 5 1 4
5 4 1 2 6 3 7
3 5 2 6 4 1 7
3 4 6 1 5 2 7
5 4 6 3 7 2 1
7 6 5 4 2 3 1
2 4 3 6 5 1 7
Appendix F. Random Rule Permutations

7  3  6  2  1  4  5
6  1  7  2  3  5  4
2  6  7  4  5  1  3
2  4  7  5  3  6  1
4  1  5  7  6  2  3
5  4  2  6  1  3  7
4  7  6  3  5  2  1
6  7  1  3  2  5  4
4  2  1  5  3  6  7
6  3  4  2  1  5  7
3  7  5  2  6  4  1
2  4  6  3  7  5  1
1  7  2  4  3  5  6
6  7  5  2  3  4  1
3  4  5  6  7  2  1
6  3  4  2  5  7  1
3  7  2  5  6  4  1
3  2  1  6  4  5  7
1  4  5  7  3  2  6
3  1  2  5  6  4  7
1  4  6  2  7  3  5
7  5  6  2  1  3  4
7  2  6  3  1  4  5
3  7  6  1  2  4  5
6  4  1  2  7  5  3
2  3  7  4  6  5  1
2  5  4  3  1  7  6
3  7  2  4  5  1  6
Appendix F. Random Rule Permutations

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
### Appendix F. Random Rule Permutations

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>3</th>
<th>6</th>
<th>2</th>
<th>4</th>
<th>5</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
References


References


