An earlier version of this paper appears in: H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood, 1992.

METAGAME: A New Challenge for Games and Learning

Barney Pell[1]
University of Cambridge
Cambridge, UK
E-mail: bdp@cl.cam.ac.uk

Abstract

In most current approaches to Computer Game-Playing, including those employing some form of machine learning, the game analysis mainly is performed by humans. Thus, we are sidestepping largely the interesting (and difficult) questions. Human analysis also makes it difficult to evaluate the generality and applicability of different approaches.

To address these problems, we introduce a new challenge: Metagame. The idea is to write programs which take as input the rules of a set of *new* games within a pre-specified class, generated by a program which is publicly available. The programs compete against each other in many matches on each new game, and they can then be evaluated based on their overall performance and improvement through experience.

This paper discusses the goals, research areas, and general concerns for the idea of Metagame.

[1]Parts of this work have been supported by RIACS, NASA Ames Research Center [FIA], and a British Marshall Scholarship.

1 Introduction

One reason why game-playing is an exciting activity for humans is that it couples intellectual activity with direct competition: better thinking and learning generally implies winning more games. Thus we can test out and refine our intellectual skills by playing games against opponents, and evaluate our progress based on the results of the competition.

The same motivation accounts for much of the original interest in Computer Game-Playing (CGP) as a problem for Artificial Intelligence: programs which think better, may play better, and so win more games. Thus we can test out and refine different theories of intelligence by writing game-playing programs which embody these different theories, and then play the programs against each other, and consider the more intelligent program to be the one which wins the most games. In the discussion which follows, we shall call this link between winning games and presumed intelligent behaviour the *competitive performance metric for intelligence*.

Unfortunately, most current approaches to CGP, including those employing some forms of machine learning, rely on the existence of a great degree of previous human analysis of particular games. This means that the human researchers usually wind up doing most of the interesting game analysis, and makes it difficult to evaluate the generality and applicability of different approaches. In short, the fact that a program wins most of its games is not actually evidence that the program (and not its programmer) is doing much interesting from an AI perspective.

To encourage researchers to address the more general issues in game-playing, while maintaining the competitive performance metric which makes CGP so attractive as a research problem, we introduce a new challenge: Metagame. The idea is to write programs which take as input the rules of a set of *new* games within a pre-specified class. The programs compete against each other in many matches on each new game, and the programs can then be evaluated based on their overall performance and improvement through experience.

Besides being a more general challenge, Metagame presents opportunities for addressing many interesting research issues in games and learning, including opponent modelling, incomplete and imperfect information, utility of computation, change of representation, strategic analysis, learning from experience, and discovery.

A companion paper ([Pel92]) presents a definition and generator for a specific class of games, called *symmetric chess-like games*, and discusses an example game produced by the generator.

The rest of this paper elaborates on the issues presented above. Section 2 substantiates the claim that CGP is too specialised, and discusses why this is undesirable. Section 3 introduces the Metagame idea, and Section 4 presents some interesting research issues possible within the context of Metagame.

2 The Gamer's Dilemma

Before discussing the assumptions behind particular methods in CGP, I illustrate what I mean by the terms *specialisation* and *game-specific*, and explain why these qualities may be undesirable. I begin with a thought experiment which I shall call *The Gamer's Dilemma*:

> Suppose that a researcher is informed that he will soon be given the *rules* of a game, G, played by a group of humans and/or programs. They all are considered to play G at a high performance level. The researcher is given a fixed amount of time to produce a program which will compete against random members of the group (the researcher is not be allowed to communicate with them beforehand). Finally, the researcher will be paid based on the number of games the program managed to win (or draw) out of some prespecified total amount of games. The researcher's goal is, of course, to maximise the money he expects to receive from his program's play.

As practitioners in the field of Computer Game-Playing, the question now is: what advice might we give to assist this hypothetical researcher? Possible answers to this question reveal the degree of specialisation inherent in existing game-playing methods. The more general advice we can give, the more general are our methods.

2.1 Game-Dependent Knowledge in Current Approaches

In what follows, we sketch the advice which might be offered from three main approaches to CGP: knowledge engineering, database enumeration and machine learning.[2]

2.1.1 Knowledge Engineering

Knowledge engineering approaches can be broken into game-specific approaches, game-tree search, and knowledge-based search.

Game-Specific Engineering Approach The most specialised answer possible would be to enumerate a list of games and current champion programs which play these games. This list might contain the following advice:

> If the game is Chess, then use the latest version of Deep Thought.

[2]Ideally, we would like to hand over a *general game-playing program*, which could study any game for a while on its own (to use its time wisely), and which would become an expert shortly after having encountered this group of players. Although it is unlikely we will ever have a program which is *totally* general, it is useful to bear this ideal goal in mind.

If the game given to the researcher happens to be on this list, this advice would be extremely helpful. Unfortunately, if the game falls outside the list, this advice would be of little use. It is becoming a common perception in Artificial Intelligence that such a list of advice is all that many researchers in CGP have to offer.[3]

Game-Tree Search An answer which more accurately reflects the lessons learned by current approaches advocates the use of a *brute-force* search method (e.g., *minimax* with $\alpha\beta$ pruning and singular extensions), combined with extremely fast routines for updating board positions. Although this technique has proven effective on several games, it presupposes that the researcher has a good *evaluation function*, which requires specific knowledge of the game.

The burden on game analysis thus shifts to the researcher, who must choose an appropriate set of features and weights for this function. Although there are some general approaches to learning weights (discussed in Section 2.3), this approach has offered very little explicit advice about the construction of appropriate features. However, we are now beginning to understand the importance of some features which may be essential in a variety of games, such as *mobility* ([Don92, Har87, LM88]).

Knowledge-Based Search We have learned that exhaustive search may not be appropriate for all games. Therefore we may also advise our researcher as follows: first, find some human who can analyse the game at expert level, then determine an appropriate set of goals and subgoals, and priorities for these goals, and finally write a knowledge-based search program which exploits these ([Wil82]). But as in game-tree search, we really have never said much explicitly about how to find useful subgoals, so again the researcher must do the difficult game analysis on his own.

2.2 Database Enumeration

While knowledge engineering approaches rely on human analysis by definition, it might seem much less so when programs construct their own database by enumerating a large set of possible positions. On the contrary, the human researcher must perform an extensive analysis of a game to determine at least the following ([AvdMvdH91, Roy90]):

- How to enumerate positions systematically?
- How to avoid generating impossible and symmetric positions?
- Given the above, are there *enough resources* to solve the problem?

By the time the human has answered these questions, it could well be argued again that much of the real work has already been done.

[3]The theme of chess-engineering in AI is discussed further in a panel at IJCAI–91 ([LHM+91]).

2.3 Machine Learning

Recently, several researchers have investigated the problem of developing computer programs which could learn how to play games ([Len83, Tad89, Eps91, CBKF91, LM88, LS91, CFR91]). However, this research is difficult to evaluate, and even more difficult to use for learning in different games, because the games, representations, learning methods, and amount of knowledge engineering vary with each researcher.[4]

In addition, most learning methods are designed to be improved based on watching or playing against *knowledgeable* opponents (e.g., [Tad89, Eps91, CBKF91, LM88, LS91, CFR91]). Although it is certainly important to understand how a program (or person) could learn from good players, it is equally important to know *how* those good players became good in the first place. Until we have an understanding of *discovery* ([Len83]), progress in machine learning will not save us from having to preface advice to other researchers with the statement: "first, find a human expert."

An example from chess may illustrate this problem. It is well known that obtaining a *passed pawn* may increase one's winning chances in some positions. While the knowledge engineering approach would build this knowledge in, a machine learning approach would have a program discover this, presumably by one player in the game creating a passed pawn, promoting it to a queen, and going on to win the game. But since it requires many careful moves to achieve the promotion of a pawn, this will generally not happen unless one player is actively trying to do so. Now, two *lazy* learning programs competing against each other will each try to gain their knowledge from their opponent. But since neither one has this knowledge yet, they are unlikely to observe it in practice, so neither is likely to learn this concept. Thus neither program will become even reasonably good at chess. This only serves to show that learning approaches which depend on better players to show the way do not address the important issue of knowledge origins, which *must* be addressed in order to develop good game-players without relying on humans to do the real analysis.

2.4 Why Specialisation is Bad

To summarise, focussing on particular games can be disadvantageous for the following reasons:

- **Labour:** Much human effort is needed each time we develop a program to play a different game, with limited advice on the real problems.
- **Generalisation:** It is difficult to say what we have learned from our research, beyond performance on particular games.

[4] Flann ([FD89]) discusses the "fixed representation trick", in which many developers of learning systems spend much of their time finding a representation of the game which will allow their systems to learn how to play.

- **Evaluation:** It is difficult to evaluate research. Playing a particular game well often means that the researcher, and not the program, has analysed the game well. Conversely, a program which does a more general analysis may not play well against highly-specialised machines.
- **Rule Analysis:** We can write successful programs, even learning programs, without understanding the ability actually to analyse games, possibly the most interesting issue in game-playing, from an AI perspective.

Thus, despite our being a field full of experts on getting computers to play games, and having developed world-champion-level game-specific programs, we are forced to leave most of the real game analysis to be done by the human researcher, and not by the computer program.

3 A More General Challenge: METAGAME

To summarise, what we would really like to have is some kind of game for which we can be more certain that playing it well really was evidence of more intelligent processing. Then we could once again justify the use of performance in competition as a way of evaluating good work.

This motivates the idea of Metagame: we shift the problem from building a program to play a *particular* game, to building one which can play any game within a *class* of games. The performance criterion is still competition: all programs would eventually compete in a *tournament*, where they would be provided with a set of games from this class, as produced by a *game generator* for this class. The programs would play many contests in each of these new games against each other, and the winner would be the one which has won the most contests by the end of the tournament.

Although the abstract idea of Metagame is very simple, a fair amount of work must be done to develop a concrete problem that can actually be addressed. First, we need to define an appropriate *class* of games. Second, we need a *communications protocol* through which players can communicate their moves. Third, we need a *game generator* which produces new instances in this class. Finally, we need to determine *resource bounds* on programs in competition. The following sections provide some general ideas for addressing each of these issues. A companion paper ([Pel92]) presents a definition, communications protocol, and generator for a specific class of games, called *symmetric chess-like games*, and discusses a new game produced by the generator, called *Turncoat-Chess*.

3.1 Class Definition

Defining a class of games requires deciding what kinds of games will be defined, and how individual games within a class will actually be represented.

3.1.1 Desiderata for a Class of Games

Many different variants of Metagame can be played, depending on what class of games it is based upon. Here we state a few desiderata for classes of games:

- **Coverage:** A good class should be large enough to include several games actually played by humans. This encourages us to generalise the lessons we learned from working on the specific games included.
- **Diversity:** In addition to known games, a class should be diverse enough to include arbitrarily many possibly different games, to discourage researchers from exhaustively analysing each possible game and still building their own analysis into the program.
- **Structure:** A class should still be small enough to represent the structure which at first blush makes the individual games appear similar. While chess-like games and trick-taking card games seem like appropriate generalisations of existing games, the class of arbitrary theorem-proving games appears to be too general.[5]
- **Varying Complexity:** The generated games should be of varying degrees and dimensions of complexity, such as *decision complexity* and *search complexity* ([AvdHH91]), so that different games afford different analysis methods. This also enables interesting experiments to test the utility of alternative methods with respect to varying degrees of complexity.

One type of game which has been studied extensively is *positional games*, in which pieces are never moved or removed once they are played. This class of games has been the domain for several general game-learning systems ([Eps91, Kof68]), and could easily serve as a Metagame class definition.

However, this class is both simple and regular (thus falling short on several desiderata), and there are well-researched games which fall outside this restricted class. In particular, it would be interesting to play Metagame based on a class complex enough to include the chess-like games, which have received much of the attention thus far in CGP. A companion paper ([Pel92]) develops the necessary components to play Metagame in the class of *symmetric chess-like games*.

3.1.2 Representation

In order to write programs which can *accept* a set of different games, we must specify how these games will be represented. Although fully-general representation languages are possible (like first-order logic or Turing Machines), it is likely that classes of games will be much more specific, especially those which can actually be produced by a generator. So, any representation language can

[5] The class of *mathematical games* ([BCG82]) is also fully general, which suggests that this class might be inappropriate for Metagame in the near future.

be used, so long as the games produced are guaranteed to be unambiguous in the chosen representation, and so long as the *semantics* corresponding to the representation is clear. A natural method of representation, pursued in ([Pel92]), is by means of a *game grammar*.

3.2 Communications Protocol

Now, assuming that we have a way of defining and generating games within a known class, there must be a *protocol* through which the playing programs can communicate. Two issues arise in this context. First, since all generated games are new, there must be a *defined language*, in which to express a player's choice of moves, which will be adequate and unambiguous for any game within the entire class. This could be either a list of state properties which are *added* and *deleted* as the effects of a move, or a more specialised and concise *move grammar*, which might be useful for recording games. In a companion paper ([Pel92]), the latter choice is adopted.

3.2.1 Humans In The Loop?

The second issue in this context is whether programs would *communicate* directly with each other, or via humans. As in the case of human or computer game generators, both options have positive and negative aspects.

The simplest option is to communicate via humans, as is the practice for most computer-game tournaments. However, a Metagame tournament is different from traditional tournaments. First, learning systems are very likely to compete, so it would be preferable for the programs to play many contests in each game type against each opponent. Second, many different new games will be played over the course of the tournament, as a variety of games provides more data. Together, these factors suggest that such a tournament would be better if a large number of contests could take place. However, if humans are required to make the moves for their programs, this could either slow the tournament down, causing fewer contests to be played, or prove very boring for the humans, causing the tournament to be less fun.

If we take humans out of the loop, this necessitates interfacing many different programs. Although not discussed in this thesis, the workbench developed as part of this research provides an interface for programs to communicate over the Internet. This still requires having a form of Ethernet to which all programs could be attached, which might be impractical given the wide range of platforms and programming languages used to develop computer games today.[6]

[6] If we do take humans out of the loop, what do they do in the meantime? One suggestion is have them play Metagame also on the new games, and the winning human can play the winning computer for a real test of human vs. machine!

3.3 Game Generator

Given a class of games, there are two ways to produce new instances within this class: we can either use human-generated games, or program-generated games.

3.3.1 Human Game Generators

The easiest way of obtaining games would be to have some humans design a set of games within the class. They must provide the games to the programs once the competition has begun, i.e., the developers of these programs can no longer help in the game-specific analysis. This procedure has the advantage that the game designers could try out their games in advance, and make sure that they are interesting and playable. However, this also has a major disadvantage, in that it forces playing-program developers to *predict* which types of games these humans will actually produce. Since human game designers may be very creative, they are an unknown variable from the perspective of scientific experiments. Thus, while we would hope that our programs could play human-generated games within a class, and we may even test our own programs against games we design, the unknown human element may cloud research issues, at least in the short term.

3.4 Programmed Game Generators

The alternative is to develop a program to generate new games within a class. If the program is transparent and available to all researchers, this has the advantage that everyone will know what kind of games to expect, which is more desirable from a research perspective.[7] However, this also has a potential disadvantage, in that the generated games may not actually be interesting. Three points may be made in connection with this concern.

3.4.1 Intelligent Game Design

First, this concern introduces an important and general issue, for which there is no simple answer. This is that *intelligent game design* is an interesting and difficult research issue in its own right. Games which survive ([AvdHH91]) do so because they provide an intellectual challenge at a level which is neither too simple to be solvable, nor too complex to be incomprehensible. Understanding the processes by which games are created and evolve, from a computational perspective, would be a valuable complement to the analysis of strategies for playing games which has been the focus of research in computer game-playing

[7] Another way of stating this is that writing programs to play any games generated by humans may result in researchers attempting to hit a moving target. A transparent generator is needed to make the problem well-defined.

thus far.[8]

3.4.2 Interestingness requires intelligence

Second, although this will matter to human observers, it will not make much difference to the programs whether they are playing interesting or boring games. In fact, if we could develop a program which, upon consideration of a particular game, declared the game to be uninteresting, this would seem to be a true sign of intelligence! So when this becomes an issue, we will know that the field has certainly matured.[9]

3.4.3 Fairness is simpler

Third, it should be possible to develop a class and generator in a manner which increases the chances that generated games within this class will at least be *fair* for both players, so that the games will be more interesting to human researchers. A companion paper ([Pel92]), attempts to achieve this goal by defining a class in which all the rules are *symmetric* between both players.

3.5 Resource Bounds

The final practical issue which must be addressed is the amount of resources which will be available to programs in competition. The most important resource, and the one which we shall consider here, is that of time constraints. First, there must be a limit on the amount of time given to programs after they are given the rules of the new games, but *before* they are forced to play the new games against opponents. Second, there must be limits on the amount of time used by programs to *play* the games, which can be a fixed amount of time per move, or per contest (or even for the entire tournament). Third, there must be a limit on the amount of time they have *between* games, so that they can reflect on their experience.

The issue of time limits is more subtle than the corresponding problem for known games, as we have no way of knowing how much time is necessary to make each move in a new game. If all programs are limited to the same amount of time, and the games are new, perhaps it does not really matter how much time they have in particular. An arbitrary suggestion which seems reasonable would be to allow 24 hours for analysis *before* competition, 20 minutes *per game* for playing, and 3 minutes for *between-game* analysis.

---

[8] The link between game-evolution and game-playing may be even tighter than we imagine. Games often change when strategic analysis has rendered them either too difficult or too boring, and each change to a game introduces new opportunities for strategic analysis, so that games and their strategies evolve together.

[9] It would be interesting to allow programs to negotiate over draws, to avoid them playing out games that neither player can win. However, this ability complicates the rules of competition, and so will be left as an idea for the future.

A concern which arises if we decide on fixed limits in *real time* is that powerful computers would have an advantage. It could be argued that a Metagame tournament is a chance to require all programs to have *equivalent* resources, as it would again be unfortunate to have clever programs defeated by brute-force programs running on vastly more powerful machines. However, limits in real time are probably unavoidable, as it would be very difficult to make this comparison across different architectures.[10] Further, the presence of powerful machines in a tournament presents the possibility for interesting confrontations between sophisticated and brute-force game-playing methods.

4 Research Areas

4.1 The Full Issues

So Metagame encourages researchers to consider the full issues behind game-playing. Because the games are possibly new, we cannot provide our programs with many ready-made features other than the rule-system defining the games and the goals. Because the programs have resource limits, and the games have grossly varying search spaces, we cannot assume that a fixed search strategy (like minimax) will necessarily be applicable. So good programs will modify their search strategies to meet the new games. Because they are not playing against experts, they cannot assume that the moves played by the winner were necessarily better, and learning systems must begin to address the problems of discovery. Further prominent issues to be addressed in this context are the tradeoff between exploration and exploitation ([Pel91]), opponent modelling, transfer of learned knowledge across games, and limited rationality. Perhaps most importantly, in this general context it would be very interesting to evaluate the tradeoff between knowledge and search: can a brute-force program still defeat knowledge-based reasoners, even in Metagame?

4.2 Incremental Progress Through Future Competitions

4.2.1 Initial Goals: Learn By Playing

Research on Metagame could also proceed in several stages. Initially, the programs would have to be able to accept formal systems and play legal, if random, games. They might at first ignore past experience, and play the same every game. Better programs might learn with respect to particular opponents in particular games, and still better ones might try to generalise across opponents, but taking peculiarities into account. Still better programs would try to generalise across different types of games. Some programs might work backward from the end, some might work forward from the start. Some might

[10] Restricting entrants to a certain class of architectures also seems impractical.

pre-compile search knowledge, while others might refine after playing. An entire spectrum of game-learning strategies could be compared and tested in a performance context of competition.

4.2.2 Eventual Prospects: Instruction

When some of the issues in learning-by-playing have become better understood, later tournaments could move on to address other themes, such as instruction and communication. For instruction, the programs could be given a set of games played before by programs. Or they could be given a hand-selected set of training examples. In either case, these would only be delivered after the program is registered in the competition, when the programmer could no longer tamper with his creation.

4.2.3 Eventual Prospects: Communication

The theme of communication of knowledge in this context is even more exciting. We could eventually allow researchers to send a *team* of programs to compete. After each game, programs in the same team would have a certain amount of time or bandwidth to communicate with each other. Programs could of course refuse to communicate, but it would be interesting to see the advantages presented by intelligent communication, even within a single tournament. Since the games would still be new to the programs (and researchers), the communication language would have to be independent of the particular game. Thus researchers would need to investigate symbolic communication (or architecture-specific communication), if they wanted to give their programs this potentially powerful competitive edge. In such a tournament, in addition to transcripts of the games, we could analyse transcripts of the team conferences, to examine the knowledge and structures which emerged and were communicated.

5 Conclusion

So, Metagame presents a framework and a test bed for addressing many interesting and important questions about game analysis, which until now has largely been performed by humans and built directly into programs. This paper has presented the general problems and concerns with this new challenge, and a companion paper ([Pel92]) addresses the specific issues involved in constructing the necessary components to play it in a particular class of games.

Leaving instruction and communication to future research, then, the next step in Metagame is to have a game-learning tournament, where programs compete against different opponents in widely-varying games, analyse new games without the help of human game-specific knowledge, learn by playing and exploring, and trade off chances of gaining useful information with chances of winning a present game.

In conclusion, meeting the challenge of Metagame may shift the field of computer game-playing back from an engineering to a research discipline, wherein winning a game would again be a sign that the program, and not simply its programmer, is doing something intelligent.

6 Acknowledgements

Thanks to Victor Allis, Nick Flann, Mike Frank, Innes Ferguson, Robert Levinson, Karl MacDorman, Dan Pehoushek, Mark Torrance, Prasad Tadepalli, and David Wilkins for interesting discussions. Thanks also to Jaap van den Herik for careful proofreading. Thanks especially to my supervisors, Steve Pulman and Manny Rayner, and to my mentor-in-absentia, Peter Cheeseman.

References

[AvdHH91] L.V. Allis, H.J. van den Herik, and I.S. Herschberg. Which games will survive. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*. Ellis Horwood, 1991.

[AvdMvdH91] L.V. Allis, M. van der Meulen, and H.J. van den Herik. Databases in awari. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*. Ellis Horwood, 1991.

[BCG82] E.R. Berlekamp, J.H. Conway, and R.K. Guy. *Winning Ways for your mathematical plays*. Academic Press, 1982.

[CBKF91] Gregg Collins, Lawrence Birnbaum, Bruce Krulwich, and Michael Freed. Plan debugging in an intentional system. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991.

[CFR91] James P. Callan, Tom E. Fawcett, and Edwina L. Rissland. Adaptive case-based reasoning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991.

[Don92] Ch. Donninger. The relation of mobility, strategy and the mean dead rabbit in chess. In H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood, 1992.

[Eps91] Susan Epstein. Deep Forks In Strategic Maps. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*. Ellis Horwood, 1991.

[FD89] Nicholas S. Flann and Thomas G. Dietterich. A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226, 1989.

[Har87] D. Hartmann. How to Extract Relevant Knowledge from Grand Master Games, part 1. *ICCA-Journal*, 10(1), March 1987.

[Kof68] Elliot B. Koffman. Learning games through pattern recognition. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(1):12–16, 1968.

[Len83] Douglas B. Lenat. The role of heuristics in learning by discovery: Three case studies. In R.S. Michalski, J.G. Carbonnel, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1. Morgan-Kaufman, 1983.

[LHM+91] Robert Levinson, Feng-Hsiung Hsu, T. Anthony Marsland, Jonathan Schaeffer, and David E. Wilkins. Panel: The role of chess in artificial intelligence research. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, 1991.

[LM88] Kai-Fu Lee and Sanjoy Mahajan. A pattern classification approach to evaluation function learning. *Artificial Intelligence*, 36:1–25, 1988.

[LS91] Robert A. Levinson and R. Snyder. Adaptive, pattern-oriented chess. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 1991.

[Pel91] Barney Pell. Exploratory Learning in the Game of GO: Initial Results. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 – The Second Computer Olympiad*. Ellis Horwood, 1991. Also appears as University of Cambridge Computer Laboratory Technical Report No. 275.

[Pel92] Barney Pell. Metagame in Symmetric, Chess-Like Games. In H.J. van den Herik and L.V. Allis, editors, *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood, 1992. Also appears as University of Cambridge Computer Laboratory Technical Report No. 277.

[Roy90] A. J. Roycroft. Expert Against Oracle. In D. Michie, editor, *Machine Intelligence 11*, pages 347–373. Ellis Horwood, Chichester, 1990.

[Tad89] Prasad Tadepalli. Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the 11th*

International Joint Conference on Artificial Intelligence, pages 694–700, 1989.

[Wil82] David E. Wilkins. Using knowledge to control tree search. *Artificial Intelligence*, 18:1–51, 1982.