To appear in *Learning in Games, 2000*

Learning to Play Expertly: A Tutorial on Hoyle

Susan L. Epstein

Abstract

Hoyle is a program that learns to play a broad variety of board games very well, in real time. Hoyle is not a traditional game playing artifact: it eschews search and tolerates, even encourages, inconsistent and incomplete knowledge. This paper describes the program's orientation and current status, reports on some of its achievements, and offers some explanation for its prowess.

1. Introduction

Hoyle is a program that plays board games. What is novel about Hoyle is that it learns to play a variety of games, learns to play them quickly, and learns to play them well. There are, to be sure, many other programs that excel at more difficult games, but each of them plays only a single game, and relatively few of them learn. Among those that do learn, learning generally requires thousands, or even millions, of contests. Hoyle, however, plays 18 different games, and typically learns to play them expertly in less than 100 contests. Moreover, Hoyle has been cognitively validated, that is, it has been shown on several simple games to learn and play much the way a person does.

The games Hoyle plays are taken primarily from anthropology books. These games are intended to capture intellectual challenges; none is as difficult as checkers, but each of them has fascinated many people for hundreds of years. Hoyle's games are played on two- and three-dimensional boards, and some of them have *stages* (periods with different rules). They range from simple games (e.g., tic-tac-toe) to games with billions of possible decision situations. Most

of them provide ample challenge for the average thoughtful game player.

The extensive literature on Hoyle chronicles its developmental phases and progress; this paper provides a current overview. It begins with a set of basic game-playing definitions, and then describes the underlying principles behind the program, the state of Hoyle's art, and the projects currently under development for it. Finally, it considers what makes Hoyle work, and why.

2. A Game-Playing Vocabulary

For clarity, let us begin with a basic vocabulary. A *game* is a noise-free, discrete space in which two or more agents (*contestants*) manipulate a finite set of objects (*playing pieces*) among a finite set of locations (the *board*). A *position* is a world state in a game; it specifies the whereabouts of each playing piece and identifies the contestant whose turn it is to act (the *mover*). Note the distinction between a *location,* a place where a single piece can reside, and a position, which describes where all the pieces currently reside.

Each game has its own finite, static set of rules for *play* (the contestants' serial behavior). These rules specify legal locations on the board, and when and how contestants may *move* (transform one position into another). The rules of a game specify its *initial state* (the starting position for play) and a set of *terminal states* where play must halt. The rules also assign to each terminal state a *game-theoretic value*, which can be thought of as a numerical score for each contestant.

The search space for a game is typically represented by a *game tree*, where each node represents a position and each link represents one move by one contestant (called a *ply*). A *contest* is a finite path in a game tree from an initial state to a terminal state. (Note the careful distinction here between a game, such as chess, and a contest at it, such as a single experience playing chess.) A contest ends at the first terminal state it reaches; it may also be terminated by the rules because a time limit has been exceeded or because a position has repeatedly occurred. The *outcome* of a contest is the value of its terminal state, or the value (typically a draw) that the rules assign to a terminated contest.

The goal of each contestant is to reach a terminal state that optimizes the game-theoretic value from its perspective. An *correct move* from position *p* is a move that creates a position with maximal value for the mover in *p*. In a terminal state, that value is determined by the rules; in a non-terminal state, it is the best result the mover can achieve if subsequent play to the end of the contest is always correct. The game-theoretic value of a non-terminal position is the best the mover can achieve from it during error-free play. If a subtree's leaves are all labeled with values, a *minimax algorithm* backs those values up, one ply at a time, selecting the best move at each node.

Retrograde analysis backs up the rule-determined values of all terminal nodes in a subtree to compute the game-theoretic value of the root. *Full retrograde analysis* minimaxes from all the terminal nodes to compute the game-theoretic value of every node in the search space, including the initial state. Full retrograde analysis encompasses the entire search tree. In contrast, *partial retrograde analysis* covers a subtree rooted at the current position; it minimaxes from only terminal nodes back up to identify the correct move from the current position. The number of nodes visited during retrograde analysis is dependent both on a game's *branching factor* (average number of legal moves from each position) and the depth of the subtree under consideration. For large game trees, such as checkers or chess, full retrograde analysis is intractable; the game-theoretic value of the initial position is unknown for any of them. (That means that one contestant or the other could have an inherent, but as yet unproved, advantage.) Indeed, unless a contest is near completion, even partial retrograde analysis in large game trees is likely to be intractable. Thus, move selection requires heuristics.

An *evaluation function* maps positions to values, from the perspective of a single contestant. A *perfect* evaluation function preserves order among all positions' game-theoretic values. For games with relatively small search spaces, one can generate a perfect evaluation function by caching the values from full retrograde analysis along with the correct moves. Alternatively, one might devise a way to compute a perfect evaluation function from a description of the position alone, given enough knowledge about the nature of the game. In this approach, a position is de-

scribed as a set of *features*, descriptive properties such as piece advantage or control of the center. It is possible, for example, to construct a perfect, feature-based evaluation function for tic-tac-toe, and thereby play without any search at all. For a challenging game, however, the identity of the features and their relative importance is unknown, even to human experts. Confronted with a large search space and without a perfect evaluation function, the typical game-playing program relies instead on a *heuristic* evaluation function that estimates the true game theoretic value of a node. Such a program searches several ply down in the tree, and minimaxes the heuristic's approximate values back up. Most classic game-playing programs devote extensive time and space to such heuristic search, or to variations on it.

Knowledge can be incorporated into a game-playing program in three standard ways. First, formulaic behavior early in play (*openings*) is prerecorded in an *opening book*. Early in a contest, the program identifies the current opening and continues it. Second, knowledge about features and their relative importance is embedded in a heuristic evaluation function. Finally, partial retrograde analysis is performed offline, to calculate and store some true game-theoretic values and correct moves. For nodes several ply from a terminal node, this is called a *closing book*. Because a heuristic evaluation function always returns any available closing book values, the larger the closing book, the more accurate the evaluation and the better a search engine is likely to perform. Hoyle takes a somewhat different approach.

3. Underlying Principles

Hoyle models the development of game-playing expertise. The rules of a game, however, are opaque to Hoyle; they serve only as an oracle. The rules provide the legal moves, and label a position as either non-terminal or, if it is terminal, provide its game-theoretic value.

Whose play to study is a separate issue. Hoyle has a variety of contestants that can compete to provide experience: external programs, people, and Hoyle. Although people may compete through a keyboard, most Hoyle experiments are now between two programs, both for consis-

tency and for speed.

Programmed competitors include Hoyle itself, as well as random, perfect, and reasonable players. A *random player* is an external program that makes random but legal moves. A *perfect player* is a handcrafted, external program for a particular game that always makes a correct move, and is equally likely to make any correct move, so that it provides a broad variety of high-quality opposition. For games with relatively small game trees, the perfect player relies on a machine-generated hash table external to Hoyle. For larger game trees the perfect player is algorithmic; such programs have typically been developed by students and then, when Hoyle learned to beat them, they are corrected and improved. To approximate other skill levels, Hoyle uses reasonable players. A *reasonable move* is an immediate win if one exists, or, if no immediate win exists, one that avoids a loss in two moves, or, if no such move exists, a random legal move. An *x%-reasonable player* is one that makes a reasonable move *x*% of the time and a move the perfect player would choose the rest of the time. A reasonable player offers variety without obvious blunders.

In some ways, Hoyle resembles a person who already plays many games well, and then learns a new game. This expert understands the general game-playing scenario (take turns, begin at some initial position, play as the rules dictate), and also knows what is important to learn about a new game, how to acquire that knowledge, and how to apply it. This remainder of this section explains Hoyle's representation and application of those ideas.

3.1 Useful knowledge

Useful knowledge is knowledge that is possibly reusable and probably correct. In game playing, for example, a good opening is a kind of useful knowledge, an *item*. Each item of useful knowledge is expected to be relevant to every game. The *values* for a particular useful knowledge item, however, are not known in advance; openings, for example, must be learned, and they will vary from one game to another. This is what is meant by *game-specific* useful knowledge.

Hoyle learns by watching competition and/or by competing itself. Hoyle is provided in ad-

vance with a set of useful knowledge items. Each item has a name (e.g., “opening”), a learning algorithm (e.g., “rote”), and a trigger (e.g., “learn after each contest”). When a useful knowledge item triggers, its learning algorithm executes, and Hoyle acquires game-specific useful knowledge. Note that there is no uniform learning method for useful knowledge items — in this sense, Hoyle is truly a multi-method learner. A learning trigger may be set for after a decision, after a contest, or after a sequence of contests.

Recall that useful knowledge is not guaranteed to be correct. Many of Hoyle’s learning algorithms are inductive, and therefore their results are subject to error. All of Hoyle’s learning is time-limited as well: the typical learning time allocation is 10 seconds per item, and a learning algorithm is terminated when it exceeds its time limit. Moreover, with several different inductive methods at work, the results may contradict each other, producing a knowledge base that is inconsistent. And, given that Hoyle learns from competition, the knowledge base may also be incomplete — 100 contests at a game is unlikely to access an entire game tree, even for tic-tac-toe.

Hoyle’s current set of useful knowledge, described below, includes symmetries, expert moves, Hoyle moves, significant states, dangerous states, and forks, as well as the Advisor weights described in Section 3.4 and some perceptually-oriented items described in Section 4. For simplicity, this discussion assumes a *draw game*, one where the game-theoretic value of the root is a draw.

A *symmetry* is a transformation of space on a game board that has no impact on any position’s game-theoretic value. Tic-tac-toe, for example, is symmetric across a horizontal or a vertical line through the center of the board. This is why, in the initial position, no tic-tac-toe corner is different from any other. Knowledge of symmetry can help Hoyle conserve resources. Not every game has the same symmetries, of course, so Hoyle learns symmetries first. Before competition begins, Hoyle observes play between a perfect player and a random player. From each non-draw contest, Hoyle extracts the final position along with the identity of the winner. To determine whether a particular symmetry is valid, the symmetry-learning algorithm transforms the board under that symmetry and applies the rules to determine what the outcome of the contest would have been

with that final position instead. If the outcome would have been the same, the symmetry is presumed to be applicable. Hoyle tests only for the standard symmetries on the two-dimensional plane: reflection across a horizontal, vertical, or diagonal line; or clockwise rotation of 90º, 180º, or 270º about the center.

An *expert move* is one an expert has made during competition in a particular position. Hoyle is told whether or not a particular contestant is expert, and simply memorizes those moves after each contest. A *Hoyle move* is one the program has made during competition in a particular position. Once again, Hoyle simply memorizes those moves at then end of each contest, along with the contest outcome. One might think that, at least for the simple games, these two useful knowledge items might occupy a good deal of space and be tantamount to memorizing the game tree. Extensive studies with Hoyle show, however, that this is not the case (Epstein 1994b). For example, in 1000 contests of tic-tac-toe between a perfect player and Hoyle, only 72 symmetrically distinct states of the possible 750 occurred. Much of high-level competition is actually quite formulaic, and deviation from it can take an expert, human or machine, by surprise. Mere memorization of the tree is a *brittle* approach, that is, one likely to fail in unfamiliar circumstances. Hoyle aspires to greater resiliency than that.

After each non-draw contest, Hoyle attempts to identify an error in the loser's play. First it identifies the *loser's last chance*, that is, the last position in which the loser had more than one non-symmetric, non-forced move. Next, it performs partial retrograde analysis from that position. If the game-theoretic value of the loser's last chance, or any non-terminal position in the searched subtree, is not a draw, the position is recorded as a *significant state* along with the correct moves from it. If Hoyle cannot complete the search within its allotted time limit, the position is recorded as a *dangerous state* instead. As Hoyle learns more, a dangerous state may eventually be transformed into a significant one, or be eliminated from memory altogether when it proves safe.

A *fork* in game playing is a position in which one contestant has more than one way to achieve a subgoal (e.g., piece capture), only one of which the other contestant can prevent on its next

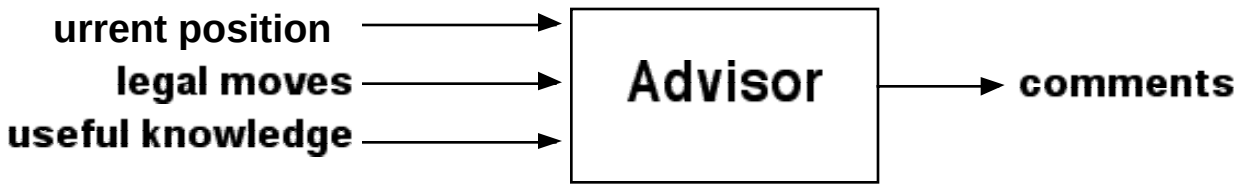

Figure 8.1: An Advisor produces comments.

move. Hoyle represents a fork as a bipartite graph (Epstein 1990). After a non-draw contest, Hoyle learns forks only if the game has *static moves*, that is, if a piece, once placed, cannot move (as in chess) or somehow be transformed (as in Othello). Hoyle applies all this useful knowledge through its Advisors.

3.2 The Advisors

An Advisor is a good reason for making a decision. Hoyle begins with a set of prespecified Advisors intended to be *game-independent*, that is, relevant to most games Hoyle plays. For example, *Victory* is an Advisor that identifies a move that will win on the next turn and comments strongly in its favor. Victory does not comment on all such moves, since any immediate win will do. Although Victory is always correct, there is no requirement that all Advisors must be, merely that they embody reasonable heuristics. For example, *Material* is an Advisor that supports capturing the opposition's pieces and opposes capture of one's own. It supports moves after which Hoyle will have more pieces or the other contestant will have fewer, and opposes those after which Hoyle will have fewer or the other contestant more. Material is often, but not always, correct.

Each of Hoyle's Advisors is a time-limited procedure that, as in Figure 8.1, accepts as input the current position, the legal moves from that position, and any useful knowledge Hoyle has acquired about the game. Until it actually plays a game, Hoyle has no values (e.g., actual openings) for a useful knowledge item; it only has the name of the item, its learning algorithm, and its trigger. As a result, an Advisor that is entirely dependent upon a particular item of useful knowledge will be unable to comment at all until some relevant values are acquired for that item during learning.

Each Advisor produces as output its opinion on any number of the current legal moves. An opinion is represented as a *comment*, of the form *<strength, move, Advisor>* where *strength* is an integer in [0, 10]. A comment expresses an Advisor's support for (*strength* > 5) or discouragement of (*strength* < 5) a particular move. Comments may vary in their strength, but an Advisor may not comment more than once on any move for the current position.

Some of Hoyle's Advisors are purely reactive, that is, they respond merely to the current position. Others, however, apply useful knowledge. For example, *Wiser* also identifies a winning move, but one that wins in more than one ply. Wiser has access to the significant states of a game, and bases its comments upon them.

Since Hoyle's Advisors vary in their reliability, they must be organized in a way that incorporates their correctness. For example, if Victory has a comment, then Material's opinion should be irrelevant. Hoyle is based upon an architecture that provides for just such contingencies.

3.3 The architecture

FORR (FOr the Right Reasons) is an architecture for learning and problem solving (Epstein 1994a). Hoyle is an implementation within FORR for the domain of game playing. Another well-known implementation is *Ariadne*, for simulated robot path finding (Epstein 1998).

FORR organizes Advisors into tiers, as in Figure 8.2, based upon their correctness and the nature of their response. In *tier 1*, FORR maintains a presequenced list of prespecified, always cor-

Table 8.1: Hoyle's tier-1 Advisors, in the order they execute.

Advisor	Interpretation
Victory	Make a move to an immediate win.
Wiser	Make a move to a winning position, based on 1-ply lookahead.
Sadder	Avoid moves to an immediate loss.
Don't Lose	Avoid moves to a losing position, based on 1-ply lookahead.
Panic	Block moves for opposition to a winning position, based on 2-ply lookahead.
Enough Rope	Leave moves for opposition to a losing position.
Shortsight	Take winning moves and avoid losing ones, based on 2-ply lookahead.

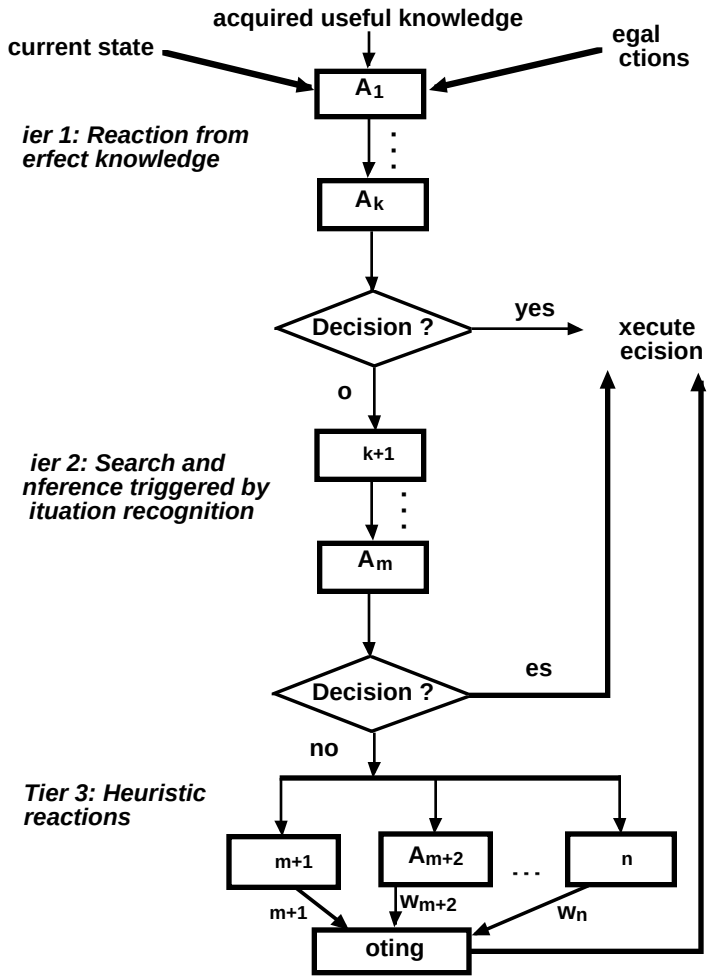

Figure 8.2: The FORR architecture organizes and manages Advisors.

rect Advisors, denoted by $A_1 \ldots A_k$ in Figure 8.2. For Hoyle, that list includes Victory, Wiser, and five others, ordered as they appear in Table 8.1. *Sadder* comments against any losing moves indicated by the significant states of a game. *Don't Lose* comments against any losing moves indicated by one-ply lookahead. *Panic* looks ahead two ply (one move for Hoyle and one for the opposition) to recommend moves that can block a loss. *Enough Rope* looks ahead the same two ply to see if the opposition might have a losing move that Hoyle could leave available, and comments against any move that would destroy that possibility (unless they all would). *Shortsight* comments in favor of winning moves, and against losing ones, based on two-ply lookahead.

Hoyle begins the decision making process at the top of tier 1 in Figure 8.2, with the current position, the legal moves from it, and any useful knowledge Hoyle has acquired about the game.

Only symmetrically distinct moves are considered. For example, from the initial position at tic-tac-toe Hoyle only considers three moves: the center, a corner, and a side square. When a tier-1 Advisor comments positively on a move, no subsequent Advisors are consulted, and the move is executed. When a tier-1 Advisor comments negatively on a move, that move is eliminated from consideration, and no subsequent Advisor may support it. If the set of possible moves is thereby reduced to a single move, that move is executed.

Typically, however, the first tier does not identify a move, and control passes to *tier 2*, denoted by $A_{k+1}…A_m$ in Figure 8.2. Tier-2 Advisors plan, and may recommend sequences of moves, instead of a single move. Tier 2 was developed for Ariadne, and is currently under construction for Hoyle.

If neither the first nor second tier produces a decision, control passes to *tier 3*, denoted by $A_{m+1}…A_n$ in Figure 8.2. Here the fallible Advisors, including Material, reside. Hoyle has an initial list of 13 prespecified, game-independent tier-3 Advisors, summarized in Table 8.2. *Anthropomorph* reproduces moves it has observed the perfect player make. Anthropomorph may be of little help during testing against an imperfect opponent, since the Advisor is applicable only to previously experienced states or their symmetric equivalents. *Cyber* reproduces Hoyle's own moves in contests that Hoyle has not lost. Cyber, too, is of no help in hitherto unvisited portions of the game tree. *Coverage* attempts to spread Hoyle's pieces across the game board. The rationale behind Coverage is that in games with predrawn straight lines on the board, such as those in Figure

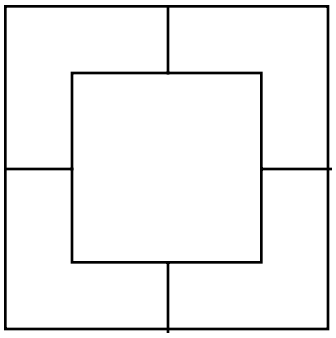

Figure 8.3: A game board with predrawn lines.

Table 8.2*:* Hoyle's tier-3 Advisors, in alphabetical order.

Advisor	Interpretation
Anthropomorph	Move as a successful expert opponent did.
Candide	Move to advance a naive plan.
Challenge	Move to maximize number of winning lines, minimize for opposition.
Coverage	Move to maximize influence on predrawn game board lines, minimize for opposition.
Cyber	Move as successful Hoyle did.
Greedy	Move to advance more than one winning line.
Leery	Avoid moves to a dangerous state.
Material	Move to increase the number of its pieces, decrease for opposition.
Freedom	Move to maximize the number of moves on next turn, minimize for opposition.
Not Again	Avoid moves previously unsuccessful for Hoyle.
Pitchfork	Advance own forks, destroy opposition forks.
Vulnerable	Avoid piece loss on two-ply lookahead.
Worried	Destroy opposition's naive offensive plans.

8.3, it is advantageous to have at least one piece on every line. Coverage encourages moves that create a new presence on a line or prevent the presence of the other contestant. *Freedom* focuses on mobility; it supports moves after which Hoyle will have more legal moves than in the current position, or the other contestant will have fewer. Similarly, Freedom opposes moves after which Hoyle will have fewer legal moves or the other contestant will have more. In games that are lost by a contestant who cannot move, Freedom may be particularly important. *Vulnerable* supports moves that, looking ahead two ply, will prevent capture of its own pieces. *Leery* opposes any move that results in a dangerous state, while *Not Again* opposes any Hoyle move that occurred during a lost contest (as recorded in Hoyle moves).

Several of Hoyle's prespecified tier-3 Advisors apply useful knowledge about forks, which includes information about *winning lines* (lines which, if completely occupied, produce a win). *Candide* supports play on any winning line for Hoyle, while *Worried* supports play on any winning line for the other contestant. *Pitchfork* supports Hoyle's use of forks and attempts to destroy those of the other contestant. In static-move games like tic-tac-toe, *Challenge* uses the fork rep-

resentation to cover as many winning lines as possible for Hoyle and minimize any opportunity for the other contestant to do the same. *Greedy* moves to advance more than one winning line on the board. Hoyle can also learn game-specific Advisors, based upon visual perception, as discussed in Section 4.

In FORR, all tier-3 Advisors are consulted in parallel, and a decision is reached by combining their comments in a process called *voting*. The move that receives the most support during voting is executed. Originally, voting was simply a tally of the comment strengths. That process, however, makes tacit assumptions that are not always correct. Instead, voting is now weighted.

3.4 Weight learning for voting

Although Hoyle begins with a set of game-independent, tier-3 Advisors, there is no reason to believe that they are all of equal significance in a particular game. Moreover, their reliability could vary from one game to another. Therefore, Hoyle uses a weight-learning algorithm called *PWL* (Probabilistic Weight Learning) to learn game-specific weights for its tier-3 Advisors. The premise behind PWL is that the past reliability of an Advisor is predictive of its future reliability.

Let the Advisors in tier 3 be $A_{m+1}, A_{m+2},\ldots, A_n$ with weights $w_{m+1}, w_{m+2},\ldots, w_n$, respectively, and denote by s_{ij} the strength of the comment from Advisor i on move j. An Advisor's weight represents the likelihood that it has, in the past, somehow supported the correct decision. At the end of each contest, whenever a true game-theoretic value is determined during search for a significant state, or whenever a perfect player makes a move, a *training example* is produced, a position and a set of one or more correct actions associated with it. These correct actions are assembled from the significant states, the learned expert moves, and any move the perfect player made. Training examples are then processed as follows.

After each contest, PWL presents each new training example to each tier-3 Advisor for comments. The weight PWL learns is the probability that the Advisor's *opinion* (set of comments produced for a given training position) is correct. In PWL the weight w_i of Advisor A_i is

$$w_i = \frac{number\ of\ correct\ opinions}{number\ of\ opinions} \quad [1]$$

Whenever an Advisor makes any comment, the denominator in [1] is incremented. To compute the numerator in [1], PWL must judge each Advisor's ability to distinguish among good and bad choices. An Advisor can *support* expertise if and only if its comment-producing algorithm can produce strengths above 5. To score positively on a training example, an Advisor that can support expertise must support correct action as highly as any other move it supports, and must discriminate, that is, like at least one other move less. (If all moves are equally good, however, any supporting comment is construed as a positive score.) Similarly, an Advisor can *oppose* expertise if and only if its comment-producing algorithm can produce strengths below 5. To score positively on a training example, an Advisor that can oppose expertise must oppose correct action as minimally as any other move it opposes, and must discriminate, that is, dislike at least one other move more. Alternatively, an Advisor that can oppose expertise also scores positively on a training example if it only opposes incorrect actions. If an Advisor can both support and oppose, it must score positively on both counts for its opinion to be correct. An Advisor that can only support, or only oppose, must score positively on its respective evaluation for its opinion to be correct.

Consider, for example, Material, Hoyle's Advisor that supports or opposes moves based on the resulting piece count. If Material recommends a correct action in a training position, and recommends no other move more strongly, PWL considers Material correct. If, however, Material prefers some other move to the correct ones, whether it comments on any correct move or not, PWL considers Material incorrect. (Failure to comment on an action is construed as a comment with strength 5, and is neither support nor opposition.) In contrast, Leery is a tier-3 Hoyle Advisor that can only oppose moves. If Leery opposes any but a correct action, it is effectively discriminating and supporting the correct one, so PWL will consider Leery correct. All action matching is normalized for symmetry.

Initially, every Advisor has a weight of .05 and a *discount factor* of .1. Each time an Advisor

comments, its discount factor is increased by .1, until, after 10 sets of comments, the discount factor reaches 1.0, where it remains. Early in an Advisor's use, its weight is the product of its learned weight and its discount factor; after 10 sets of comments, its learned weight alone is referenced. Thus, in tier 3, PWL chooses the move with the greatest support, that is,

$$\underset{j}{argmax}\left\{\sum_i \omega_i w_i s_{ij}\right\} \text{ where } \begin{cases} d_i = \text{number of opinions } i \text{ has generated} \\ \omega_i = \begin{cases} 0.1 * d_i \text{ if } d_i < 10 \\ 1 \text{ otherwise} \end{cases} \end{cases} \quad [2]$$

If an Advisor is correct, its wisdom will gradually be incorporated. If an Advisor is incorrect, its weight will diminish as its opinions are gradually introduced, so that it will have little negative impact.

During testing, PWL drops Advisors whose weights are no better than random guessing. This threshold is provided by a non-voting tier-3 Advisor called *Anything.* Anything comments only during weight learning (i.e., not for play decisions). It comments on one move 50% of the time, on two moves 25% of the time, and in general on n moves $(0.5)^n$% of the time. Each of Anything's comments has a randomly generated strength in {0, 1, 2, 3, 4, 6, 7, 8, 9, 10}. An Advisor's weight must be at least .01 greater than Anything's weight to be consulted during testing. During testing, provisional status is also eliminated (i.e., ω_i is set to 1), to permit infrequently applicable but correct Advisors to comment at full strength. In summary, PWL fits FORR to correct decisions, learning to what extent each of its tier-3 Advisors reflects true game-theoretic values. Because the perceptually-supported learning of the next section spawns Advisors of unknown merit, PWL is essential to robust learning and to perceptual enhancement.

4. Perceptual Enhancement

People play better when they can see the game board. Hoyle has two kinds of representations that attempt to capture visual perception: patterns and zones. Patterns and zones are optional

features in Hoyle, but sometimes very powerful ones. This section reviews each of them.

4.1 Patterns

Playing pieces and blanks can form many different arrangements on a game board. Even on a board as small as tic-tac-toe's, there are 4^9 distinct combinations of X's, O's, blanks, and #'s (*don't care* symbols). For efficiency, Hoyle restricts the patterns it learns to one of five basic shapes that the mammalian vision system is specialized to detect quickly: straight lines, V's, L's, squares, and the diagonals of squares.

When Hoyle first encounters a new game, it finds symmetrically distinct ways to place and size these shapes on the board. Each such way is called a *template.* Then, during play, Hoyle records the instantiation of each template as a *pattern*. A pattern forms when a move changes the contents of a template, and is thereby linked with a particular *role* (the first or second contestant).

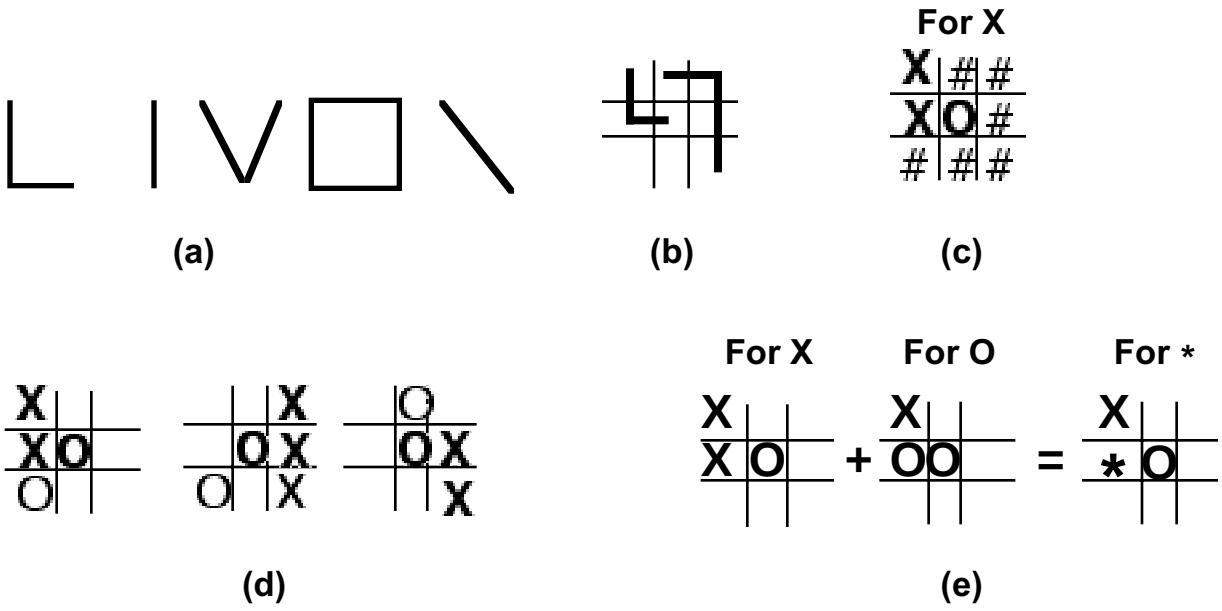

Figure 8.4: (a) Shapes used by Hoyle to detect patterns. *(b)* Two of the ways the L shape can be fitted to a 3×3 grid to form a template. *(c)* A pattern derived from the smaller L template in (b). *(d)* Some positions that match either the pattern in (c) or a symmetric equivalent of it. *(e)* A pattern generalization.

This progression from shape to template to pattern is shown in Figures 4(a) through 4(c).

Patterns that occur several times during competition are collected, and Hoyle begins to associate each of them with an outcome (win, loss, or draw). If, for example, a contestant that creates a black L in the corner of the board usually wins, Hoyle's pattern learning algorithm will record that. Patterns that *persist* (recur regularly) and have a consistent association are placed in a *pattern cache* with that association. Hoyle has a tier-3 Advisor called *Patsy* that comments in favor of moves that create patterns with positive associations, and against moves that create patterns with negative associations. For example, a move by X that led to any of the boards in Figure 8.4(d) would be said to match the pattern in Figure 8.4(c). If Figure 8.4(c) were a pattern with positive associations (winning or drawing), Patsy would support such a move; if it were a pattern with a negative association (losing) Patsy would oppose it. One move is likely to create (and, if pieces slide or are captured, destroy) more than one pattern, so Patsy computes each comment's strength based upon a move's overall effect, weighting each pattern equally.

In addition to Patsy, Hoyle spawns new, learned, game-specific, pattern-oriented Advisors after every 10 contests. To do so, it sweeps the pattern cache and generalizes over two or more patterns stored there. Figure 8.4(e) is an example of pattern generalization. Each such generalization becomes a new, game-specific, learned tier-3 Advisor, with an initial weight of .05 and a discount factor of .1. ⇒ Like Patsy, a pattern-oriented Advisor supports any move that produces a pattern with a positive association for the mover, and opposes any move that produces a pattern with a negative association. An individual pattern-oriented Advisor, however, is restricted to its single, generalized pattern and has its own learned weight.

4.2 Zones

Hoyle's second kind of perceptual enhancement views the predrawn lines and legal locations on the game board as a graph. The program distinguishes between a *placing move,* which puts a piece on the board for the first time in a contest, and a *sliding move,* which shifts a piece already on the board from one location to another immediate adjacent empty location along a predrawn

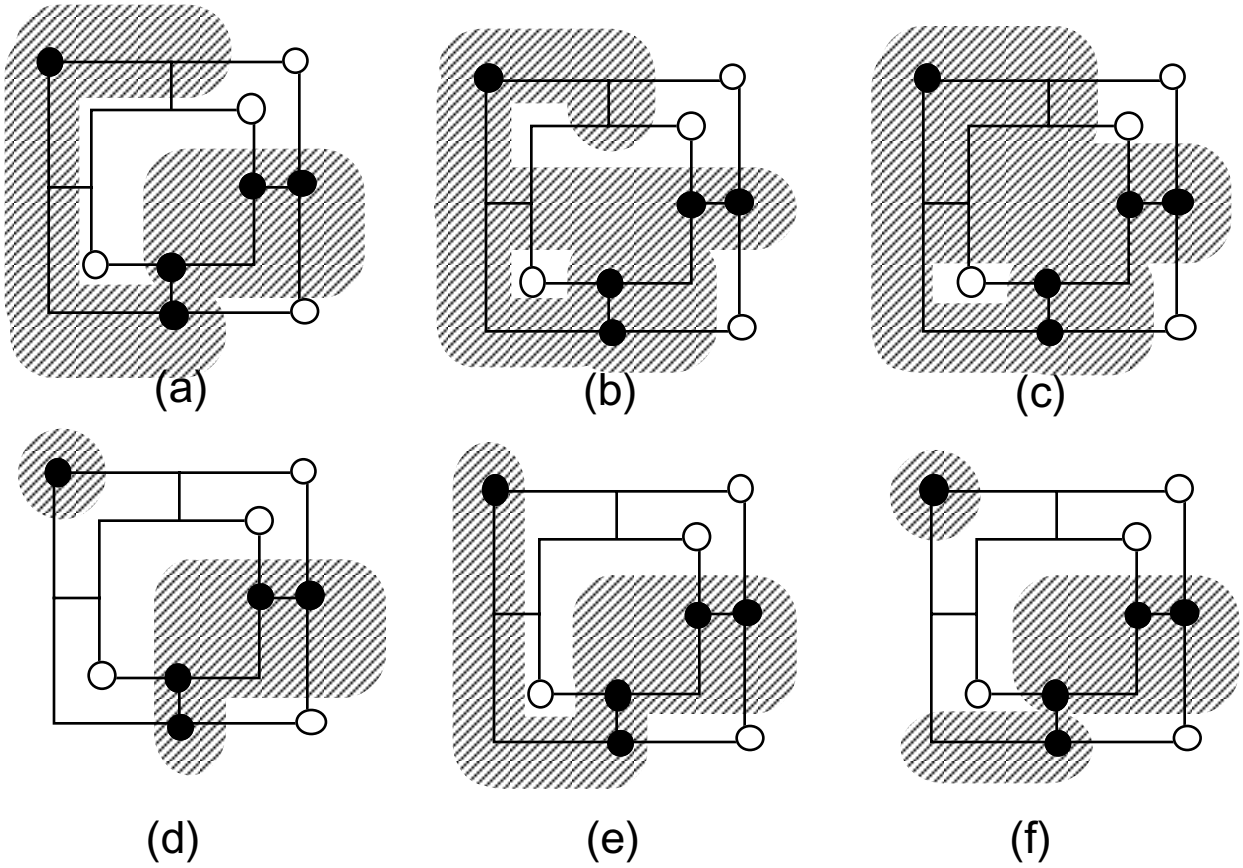

Figure 8.5*:* A black zone defined for six zone types on the same board: *(a)* one-step, *(b)* two-step, *(c)* multi-step, (*d*) piece-bounded, (*e*) private, and (*f*) extended private.

line. A *zone* is a set of locations that are *connected* by predrawn lines, so that it is possible to travel from any location in a zone to any other location in the same zone along a path through the zone. The locations in a zone are either empty or contain pieces belonging to a single contestant (the *owner* of the zone), and every zone contains at least one piece.

Hoyle has six ways to construct the boundaries of a zone, demonstrated on one board position in Figure 8.5. Intrinsic to most of these zone types is the notion of distance; the *distance* between two locations is defined here to be the number of sliding moves required to reach one from the other. In a *one-step zone* every empty location is of distance one from at least one of the owner's pieces in the zone. Black's single one-step zone is shaded in Figure 8.5(a). In a *two-step zone,* every empty location is of distance one or two from at least one of the owner's pieces in the zone. The two-step zone corresponding to Figure 8.5(a) appears in Figure 8.5(b). In a *multi-step*

zone every empty location is within a finite distance from at least one of the owner's pieces in the zone. The multi-step zone corresponding to Figure 8.5(a) appears in Figure 8.5(c). In a *piece-bounded zone,* the perimeter of the zone is controlled by owned pieces, so that, unless perimeter pieces move, the zone is impenetrable by the other contestant in any number of sliding moves. Black's single piece-bounded zone is shaded in Figure 8.5(d). In a *private zone* every empty location is of distance one from at least one of the owner's pieces in the zone and is not of distance one from any from the other contestant's pieces. The private zone corresponding to Figure 8.5(a) appears in Figure 8.5(e). Finally, in an *extended-private zone* every empty location is of distance no more than two from at least one of the owner's pieces in the zone and is not of distance one or two from any of the other contestant's pieces. Black's two private-extended zones corresponding to Figure 8.5(a) appear in Figure 8.5(f).

Hoyle recognizes eight *features* for a zone, descriptions a person might notice: the number of pieces in it, the number of (empty or occupied) locations it includes, its *density* (the ratio between the number of its pieces and the number of its locations), its *compactness* (the smaller of its height to width and width to height ratios), the number of predrawn lines it completely includes, the number of predrawn *short* (length 2) lines it completely includes, the number of predrawn *long* (length 3) lines it includes, and the maximum internal path length between any pair of locations. Compactness and path length are attempts to capture shape. The one-step zone in Figure 8.5(a) has 5 pieces, 9 locations, density 5/9, compactness 1, and maximum internal path length 8. It completely includes 3 predrawn lines, 1 long and 2 short. In contrast, the multi-step zone in Figure 8.5(c) has the same 5 pieces, but 12 locations, density 5/12, compactness 1, maximum internal path length 9, and includes 5 predrawn lines, 1 long and 4 short. Finally, since the number of zones a contestant has may be important, that was included as a feature whose value on a zone is always 1.

In any given position a contestant has a set of one or more zones, each of which returns a value for a given feature. Hoyle computes four statistics for each feature: the minimum, the maximum, the average, and the total of its values across the set of zones in a single position. For example, in

Figure 8.5(d), black's pieces lie in two piece-bounded zones, so the minimum number of pieces is 1, the maximum is 4, the average is 5/2, and the total is 5. When a game includes both placing and sliding moves, the representation notes the *stage* (placing or sliding) as well. Thus Hoyle's feature-based zone language has 6 types × 9 features × 4 statistics × 2 stages = 432 expressions.

At the end of each contest, for each position, Hoyle's zone-learning algorithm identifies the zones of each contestant and calculates their feature values. The algorithm stores each zone expression's values separately for the winner and the loser of the contest. If the contest was a draw, it stores the values for both contestants without distinction (unlike the pattern-learning algorithm, which includes the mover). Thus each zone expression is associated with three lists of values: one for wins, one for draws, and one for losses.

During learning, at the end of each contest, the zone learning algorithm computes which zone expressions are statistically significant. A zone expression has a *win attitude* if and only if its win values differ in the same direction (i.e., higher or lower) from both its draw values and its loss values in a statistically significant way at the 99% confidence level. A zone expression has a *loss attitude* if and only if its loss values differ in the same direction from both its win values and its draw values in a statistically significant way at the 99% confidence level. Note that, rather than express a particular number (e.g., "try for 3 zones"), Hoyle's representation distinguishes larger from smaller, a coarser approach. Zone expressions with either the same win attitude or the same draw attitude for at least 20 consecutive contests are said to be *active*. For efficiency, after the first $n > 25$ training contests, zone expressions whose means and standard deviations for their three lists differ by less than $.01n/10$ are relegated to *inactive* status, that is, their values are no longer computed or compared.

Zone Ranger is a tier-3 Advisor that comments based on active zone expressions, just as Patsy does for cached patterns. It supports moves that lead to positions where the value identified by a particular active zone expression moves in the direction believed to be correct. For example, if average density for one-step zones in the placing stage were such a zone expression, with higher

values observed as better, during the placing stage Zone Ranger would support moves that increased the mover's average density for one-step zones and oppose moves that decreased it. Similarly, it would oppose moves that increased the non-mover's average density for one-step zones during placing and support moves that decreased it. Since there are likely to be many active zone expressions, Zone Ranger combines all the evidence for and against each move, based on all the active zone expressions, and constructs the strengths of its comments accordingly.

To identify salient concepts and to control computation costs, it is necessary to filter out particularly important active expressions. Any active expression whose weight as an Advisor (see Section 8.3.4) would have been at least .85 is identified as *important*. Important expressions do not participate in Zone Ranger's computations, that is, they are no longer active.

Hoyle spawns new, learned, zone-oriented Advisors after every 10 contests. A zone-oriented Advisor is created for each important zone expression, with win attitude preferred to loss attitude. The Advisor includes an expression, its direction (increase or decrease), and whether it has a win or a loss attitude.

In summary, the Advisors of Table 8.3 simulate visual perception for Hoyle. All of them reside in tier 3. Two, Patsy and Zone Ranger, are prespecified and rely on useful knowledge. The other perceptual Advisors are game-specific and are created during execution. They may vary in number from one run to the next, and are determined by the experience Hoyle has during learning.

Table 8.3: Hoyle's tier-3 perceptual Advisors.

Advisor	Interpretation
Patsy	Move to a position whose patterns have positive associations for Hoyle, negative for the opposition.
Learned pattern Advisors	Move to a position with a pattern generalization that has positive associations for Hoyle, negative for the opposition.
Zone Ranger	Move to a position whose zones have positive associations for Hoyle, negative for the opposition.
Learned zone Advisors	Move to a position with a zone expression that has positive associations for Hoyle, negative for the opposition.

This makes Hoyle's empirical structure extremely important.

5. An Empirical Framework

Hoyle is used not only to learn new games but also to learn about learning environments. FORR provides Hoyle with an empirical framework that supports such careful study. The framework isolates learning from testing, establishes performance metrics, and provides for non-determinism.

The unit of performance in this framework is a contest, one complete experience at a game. A sequence of contests is called a *phase*, and may be either for testing or for learning. During a *testing phase*, Hoyle competes against an external opponent, and no learning occurs. (As a result, the program can make the same error repeatedly during testing.) During a *learning phase*, in contrast, Hoyle triggers its learning algorithms and applies them as described earlier. A learning phase may be specified either as a *watching phase*, where Hoyle merely observes two competitors, or a *doing phase*, where Hoyle takes the part of at least one contestant. (Recall that Hoyle can play against itself.) A *run* is a sequence of learning phases, followed by a sequence of testing phases. Competitors are always selected from the programs described in Section 8.3.

The fundamental performance metrics for Hoyle are speed and accuracy. The program's decision speed is measured in each phase, as well as its *power* (percentage of contests won) and its *reliability* (percentage of contests won or drawn). Since a human expert plays well not only against other experts, but against competitors of every caliber, Hoyle's skill is tested at several levels. A testing phase is typically 10 contests against a perfect player, 10 contests against a strong player (modeled by a 10% reasonable player), 10 contests against a novice (modeled by a 50% reasonable player), and 10 contests against a random player. This kind of testing renders mere memorization of the training positions insufficient for good performance. In two games defined in the next section, for example, 51.48% of the lose tic-tac-toe testing positions were never seen during training; in five men's morris that number rose to 75.55%.

Table 8.4: Hoyle's performance on 3 games. Figures in boldface are different from those in the preceding line at the 95% confidence level. Optima are computed from 1000 contests.

Versus	**Perfect**	**Strong Player**		**Novice Player**		**Random Player**	
	Reliability	**Reliability**	**Power**	**Reliability**	**Power**	**Reliability**	**Power**
Tic-tac-toe	100.0	100.0	25.0	100.0	64.0	100.0	94.0
optima	*100.0*	*100.0*	*15.3*	*100.0*	*56.1*	*100.0*	*87.7*
Lose tic-tac-toe	97.0	93.0	15.0	92.0	47.0	95.0	74.0
Lose tic-tac-toe with patterns	**100.0**	**99.0**	16.0	**100.0**	**65.0**	99.0	76.0
optima	*100.0*	*100.0*	*17.5*	*100.0*	*60.4*	*100.0*	*78.6*
5 men's morris	83.0	82.0	12.0	83.0	53.0	96.0	80.0
5 men's morris with zones	**97.0**	91.0	**0.0**	90.0	**27.0**	95.0	**57.0**
optima	*100.0*	*100.0*	*31.3*	*100.0*	*89.2*	*100.0*	*99.6*

Because tier-3 voting ties are broken at random, and because many of the competitors select from some set of choices at random, Hoyle is a non-deterministic program. To measure its skill accurately, therefore, performance is averaged over a set of runs, called an *experiment*. Thus one defines an experiment with Hoyle by specifying a number of runs, each with the same structure. A sample experiment might be described as follows: in each of 10 runs, have Hoyle watch the perfect player compete against itself for 20 contests, then have Hoyle compete against the perfect player for 80 contests, and then turn learning off and test for 10 contests against the perfect player, against a 10%-reasonable player, against a 50%-reasonable player, and against a random player.

6. Results

This section focuses on Hoyle's behavior with respect to the performance standards of the preceding section: speed and prowess. The interested reader is referred to (Ratterman and Epstein 1995) for a demonstration of Hoyle's cognitive plausibility and to (Epstein 1994b; Epstein 1995)

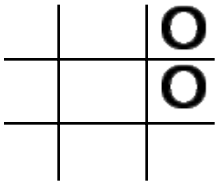

Figure 8.6: Hoyle learns this significant state in tic-tac-toe on 9 out of 10 runs, along with a forking move guaranteed to win from it.

for discussions of the impact of the training environment.

At this writing, Hoyle has learned to play 18 games expertly. The data in Table 8.4 from three games is representative. Because these are all draw games, power against the perfect player is always zero, and has been omitted. For comparison, data for optimal play (averaged over 1000 contests between the perfect player and the same testing programs) is provided as well.

As Table 8.4 indicates, Hoyle learns to play flawless tic-tac-toe after a learning phase that pits it against the perfect player for 100 contests, without any perceptual learning. Hoyle only lost 7 contests in all 10 training runs, and only on one run did it lose two. In the process Hoyle tunes its weights, and acquires a small amount of useful knowledge, itemized later in this section. Figure 8.6 provides an example.

At *lose tic-tac-toe*, a game played exactly like tic-tac-toe except that three in a row, column, or diagonal loses, matters are somewhat different. This game has been solved mathematically, with theorems and proofs that demonstrate differences between the correct strategies for the two players (Cohen 1972). An intrinsic part of that paper's discussion is the notion of reflection of the previous move across the center of the board. Although this is by no means enough to guarantee perfect play, it is certainly important. After Hoyle plays 100 contests of lose tic-tac-toe against the perfect player, the program still does not play perfectly reliably. Its knowledge base is too low-level; it contains details about specific positions, but not the principles behind them. Although that is good enough for tic-tac-toe, for lose tic-tac-toe it is not. When pattern learning is enabled, however, Hoyle achieves near-perfect reliability, losing only two testing contests in 10 runs. Note the improvement in performance from the third to the fourth rows of Table 8.4.

Table 8.5: Hoyle's decision time in seconds per contest during testing against 4 competitors.

Game	Perfect player	Strong player	Novice player	Random player
Tic-tac-toe	0.19	0.18	0.16	0.14
Lose tic-tac-toe	0.19	0.18	0.19	0.19
Lose tic-tac-toe with patterns	0.38	0.37	0.33	0.35
5 men's morris	22.31	21.96	19.76	16.19
5 men's morris with zones	25.26	25.05	21.79	17.54

Five men's morris is played on the game board of Figure 8.5 between black and white, each of whom has five pieces. The game consists of a *placing stage*, where a move places a piece on an empty location, and a *sliding stage*, where a move slides a piece to an immediately adjacent, empty location along a predrawn line on the game board. Three same-color pieces along a predrawn line is called a *mill.* In either stage, if a move creates (or recreates) a mill, the mover also permanently captures one opposition piece. (The captured piece may not be in a mill, unless all opposition pieces are in mills.) Play terminates when the mover is reduced to two pieces or cannot slide (the non-mover is declared the winner) or when the same position recurs for the third time (a draw) or a contest has gone to 80 moves (a draw). Five men's morris has a substantial game tree (about 7 million positions), and most people find it quite challenging. Hoyle learns to play five men's morris best if it alternately observes the perfect player against a variety of opponents and plays them itself, 2 at a time against a perfect player, and against 10%, 25%, 50%, and 100% reasonable challengers, for a total of 40 contests, and then observes the perfect player against itself for 20 contests. After watching only 60 contests, Hoyle learns to play surprisingly well; enhanced with zones it learns to play even better. (See the sixth and seventh rows of Table 8.4.)

Despite its prowess, Hoyle is frugal with both time and space. It makes decisions in real time. Table 8.5 gives Hoyle's average decision time during testing against each of its competitors. Observe the increased computation time with perceptual enhancement. The storage space Hoyle requires for its useful knowledge is surprisingly small, however, and perceptual enhancement requires little additional memory. As Table 8.6, indicates, although tic-tac-toe has 750 distinct asymmetric positions, Hoyle learns very few of them. Lose tic-tac-toe, despite the same size search space, requires a good deal more useful knowledge. People find this game far more difficult than ordinary tic-tac-toe, and Hoyle does too (Ratterman and Epstein 1995).

Current research with Hoyle addresses planning, forgetting, and teaching. Hoyle has begun to learn tier-2 Advisors that propose *plans* (sequences of actions) for a specific game. FORR provides a structure that interleaves plans with single-action comments, and replans as necessary. Recent work on forgetting addresses the slowdown observable in Table 8.5. A small transposition table seems likely to improve timing. Finally, work is planned on transforming Hoyle the learner into Hoyle the teacher, so that people might learn to play as well as Hoyle.

7. Conclusion: Why Hoyle Works

Some AI researchers are puzzled by Hoyle. Impressed by its versatility, they still wonder why it succeeds without large knowledge bases or deep search. This section attempts an explanation.

Table 8.6: Hoyle's space requirements during testing.

Game	Hoyle moves	Expert moves	Dangerous states	Significant states	Patterns or zones	New Advisors
Tic-tac-toe	8.8	37.1	0.0	42.8	—	—
Lose tic-tac-toe	14.6	40.2	0.6	160.6	—	—
Lose tic-tac-toe with patterns	14.5	40.4	0.4	182.5	67.3	4.2
5 men's morris	62.1	600.9	0.3	44.3	—	—
5 men's morris with zones	57.1	600.7	0.7	39.2	221.5	19.9

Hoyle is really an expert game player; it just begins without any knowledge about a particular game. The Advisors that do not rely on useful knowledge are good heuristics. Although such an Advisor may not apply to every game, it must be applicable to a family of games before it is considered for implementation. For example, Material applies to games where pieces are captured. It also comments in tic-tac-toe, although not in a helpful manner, so PWL discards it. Different Advisors are more important in different games. Without perceptual enhancement, the high-weight Advisors for tic-tac-toe are Cyber and Anthropomorph; for lose tic-tac-toe they are Anthropomorph, Cyber, and Leery; and for five men's morris they are Material, Vulnerable, Anthropomorph, Freedom, and Cyber. (Recall that Hoyle plays 18 different games, so these preferences are only a sample.)

Hoyle works, in part, because it already knows what to learn and how to learn it. The prespecified Advisors are general game-playing principles. Hoyle knows which Advisors belong in which tier, and knows the order for the tier-1 Advisors in advance. Hoyle knows what the relevant useful knowledge items are, and has a learning algorithm for each of them.

Another key to Hoyle's prowess is its reliance on satisficing. An adequate decision can often be made without exhaustive search. Tier 1 protects Hoyle from obvious blunders and supports obvious success. After that, a consensus among good reasons for a move is usually good enough. Given the amount of time it has to consider a decision, Hoyle makes a reasonable choice.

Hoyle also recognizes that weights are crucial to accurate decision making, and has a reliable way to generate training examples for PWL. The weight learning algorithm helps Hoyle establish a synergy among good reasons for choosing a move. People familiar with more traditional game playing programs often ask about a feature-based evaluation function. Hoyle's decision routine is somewhat more elaborate, although the voting in tier 3 might be considered a dynamic evaluation function where both the weights and many of the features are learned. If an Advisor references useful knowledge, not only its weight but also the "value of the feature" (the Advisor's comments) on the same position is likely to change as training progresses. PWL's discount factor is also essential. It makes certain that new, learned Advisors do not disrupt any expertise that

Hoyle may have developed before they were introduced.

Still another part of the explanation is visual perception, which plays an important role in human expertise. Visual perception focuses our attention, cues our memories, and narrows our search. As we develop expertise, people not only identify, organize, and filter perceptually-based information as it arises, but also learn to integrate it into high-level decision making. With its patterns and zones, Hoyle is a pioneer in this *perceptually-supported learning.* Hoyle's perceptual enhancement provides it with generalizations (patterns, zones, and learned Advisors) that provide reliable guidance when Hoyle encounters unfamiliar positions. For example, Zone Ranger offers advice in 98.69% of the five men's morris testing positions, compared to Anthropomorph's 32.03% and Material's 7.00%. Similarly, at least two pattern Advisors comment more frequently than Anthropomorph during every testing phase in lose tic-tac-toe.

In summary, Hoyle works because it knows much about game playing in general, it satisfices, and it capitalizes on its experience. As a result, Hoyle learns many games quickly and learns to play expertly in real time. Its reasoning process is transparent, its learned knowledge on target, and its performance robust. As a result, Hoyle continues to serve as a test-bed for learning experiments, and a developmental framework for machine learning.

References

Cohen, D. I. A. 1972. The Solution of a Simple Game. *Mathematics Magazine,* 45(4): 213-216.

Epstein, S. L. 1990. Learning Plans for Competitive Domains. In *Proceedings of the Seventh International Conference on Machine Learning,* 190-197. Austin: Morgan Kaufmann.

Epstein, S. L. 1994a. For the Right Reasons: The FORR Architecture for Learning in a Skill Domain. *Cognitive Science,* 18(3): 479-511.

Epstein, S. L. 1994b. Toward an Ideal Trainer. *Machine Learning,* 15(3): 251-277.

Epstein, S. L. 1995. Learning in the Right Places. *Journal of the Learning Sciences*, 4(3): 281-319.

Epstein, S. L. 1998. Pragmatic Navigation: Reactivity, Heuristics, and Search. *Artificial Intelligence*, 100(1-2): 275-322.

Ratterman, M. J. and Epstein, S. L. 1995. Skilled like a Person: A Comparison of Human and Computer Game Playing. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society,* 709-714. Pittsburgh: Lawrence Erlbaum Associates.