This paper appeared in 1992 in *Minds and Machines*, 2 : 239-265.

The Role of Memory and Concepts in Learning

SUSAN L. EPSTEIN

Department of Computer Science

Hunter College and The Graduate School of The City University of New York

Abstract. The extent to which concepts, memory, and planning are necessary to the simulation of intelligent behavior is a fundamental philosophical issue in Artificial Intelligence. An active and productive segment of the AI community has taken the position that multiple low-level agents, properly organized, can account for high-level behavior. Empirical research on these questions with fully operational systems has been restricted to mobile robots that do simple tasks. This paper recounts experiments with Hoyle, a system in a cerebral, rather than a physical, domain. The program learns to perform well and quickly, often outpacing its human creators at two-person, perfect information board games. Hoyle demonstrates that a surprising amount of intelligent behavior can be treated as if it were situation-determined, that often planning is unnecessary, and that the memory required to support this learning is minimal. Concepts, however, are crucial to this reactive program's ability to learn and perform.

Key words. Representation, cognitive architecture, concepts, machine learning, game playing.

The thesis of this paper is that reactive, hierarchical systems can minimize deliberation, but that both memory and explicitly represented concepts are necessary if a program is to learn to perform intelligently. Some preliminary definitions and an overview follow.

Learning is when subsequent behavior is transformed by previous (execution) experience. *Concept* is defined here as some recognized set of regularities detected in some observed world.[1] *Regularity* means repeated occurrence and/or consistency of use. For this research, a concept includes not only the necessary and sufficient descriptions called definitions, but also the defaults, associations, and expectations in what is sometimes termed a conception. Thus a concept may incorporate error, bias, and inconsistency. The definition of a concept, even one so deceptively simple as "apple," is lengthy and non-trivial (e.g., Wierzbicka, 1985).

From an AI perspective, a concept is generalized domain knowledge, a description of what has been encountered. Although specific examples may be remembered, a concept is not a set of instances but a summary of experience. Observation of a regularity, such as "all ravens ever experienced are black," may result in an inductive generalization, such as "all ravens are black." A pragmatic response to the philosophical dilemma "What can be inductively determined?" is the representation of a concept as a set of questions with procedures to compute their answers. Of course one could be drawn into the circularity argument, that the presumption of a class of experiences in which to observe regularity of itself presumes that the concept, i.e., the class, already exists. To avoid this problem, this paper assumes that the concept preexists but is incompletely specified. As Goodman wrote, "…agreement with regularities in what has been observed is a function of our linguistic practices" (1973). Thus assume that one can recognize ravens and know about color, but will learn that they are black.

If a machine is constructed to meet a goal, the claim here that either implicit or explicit concept representation is necessary. The construction of a cherry pitter, for example, implicitly references the concept of a cherry as a small, round object containing an even smaller object which can be extracted when pressure is appropriately applied. Although the architecture of a sufficiently elaborate machine, like a robot, may obscure its concepts, this paper contends that they are present implicitly, in circuitry and mechanical devices. Thus any program claimed to be "representation-free" is characterized here as a program with implicit concept representation. In contrast, explicit concept representation makes facts and beliefs immediately accessible without need for induction or deduction. This paper demonstrates three benefits of explicit, rather than implicit, representation for a machine that learns: organization of knowledge, focus of attention, and ability to discard (ignore or forget) experience.

The first section of this paper describes a significant philosophical controversy in AI over the role of explicit concept representation in the simulation of intelligent behavior, what is termed here the *control-concept controversy* (Brooks, 1991; Kirsh, 1991). It also includes some careful definitions and a historical overview. The second section describes four programs that learn to play simple games well. The programs rely upon prespecified implicit concepts, and perfect play does not come readily to them. They have surprisingly large memory requirements, and those with more sophisticated concepts learn to play faster and better. Section 3 characterizes four kinds of explicit concepts worth learning, and summarizes the significance cultural psychology has assigned to explicit concepts in human behavior and in human expertise. It also propounds a philosophy about general problem solving, what is important to learn, and how that knowledge can be applied. The fourth section describes Hoyle,[2] a program that learns to play many games expertly. Section 5 argues that Hoyle qualifies as reactive, and addresses the origin of low-level behavior. The sixth section sketches how and why Hoyle uses concepts and memory to learn to simulate intelligent play, and demonstrates the degradation of Hoyle's performance when memory and/or explicit concepts are removed. The final section indicates how this process is expected to scale up and the impact of this work on the control-concept controversy.

The contribution of this paper is its demonstration of how explicit, rather than implicit, concept representation strengthens a reactive system that learns and reduces its reliance on memory. The control-concept controversy is addressed within a broad and challenging domain that meets the criteria previously cited as inhospitable for a reactive system: two-person, perfect information board games (Kirsh, 1991).[3] The argument uses several programs, including Hoyle, to demonstrate how a reactive system that learns to play such games can have unreasonable memory requirements. It goes on to show that explicit concept representation alleviates this problem and improves performance while preserving the essential features of a reactive system: refusal to plan, reluctance to search, and reliance on low-level responses to achieve high-level goals.

1. The Issue

This section traces the development of the consensus that a general machine intelligence must be able to learn domain-specific knowledge. Whether such a system need only learn *control* (how to behave) or must also learn concepts (generalize its experience) is the open question formulated here. Reactive systems are characterized in this section and additional relevant definitions are provided.

People have long been intrigued with the prospect of a general, machine-based intelligence. One of the highlights of early AI research was a general problem-solving system called GPS (Newell et al., 1963). It relied on a single method, *means-ends analysis*, to work in any domain. Although GPS was able to perform well on some problems, empirical investigations indicated that it often needed problem-dependent information to achieve success. The classic *generality vs. power* tradeoff in AI was now identified: so-called *weak methods* like means-end analysis could address a broad range of problems, but inefficiently and without insight. Strong methods, it appeared, would have to be specialized to incorporate knowledge about the task at hand.

In its thirty-odd years, AI has studied several ways to improve weak methods with domain knowledge. A good description of a problem (*representation*) can simplify and clarify its solution, as DENDRAL's did (Lindsay et al., 1980). A perfect representation actually makes solution trivial, i.e., search-free (e.g., Amarel, 1968). Without a perfect representation, domain knowledge can guide the exploration of alternative solutions (*search control*) as MYCIN's did (Shortliffe, 1976). Domain knowledge can also indicate which ideas are particularly relevant to the problem's solution and which are not (*focus of attention*), as ABSTRIPS and AM did (Sacerdoti, 1974; Lenat, 1976).

It is not always possible, however, for people to identify key domain knowledge precisely for an implementation. Alternatively, a program with a general method that discovered the right problem-solving knowledge could specialize itself sufficiently to develop adequate power. *Machine learning* architectures like SOAR, PRODIGY, and THEO are based on this premise (Laird et al., 1987; Minton, 1988; Mitchell et al., 1989). They are said to learn because their behavior on both previously encountered and related problems is directly affected by their prior problem-solving experience. SOAR, PRODIGY, and THEO are all based upon the premise that knowledge strengthens performance, and that such knowledge can be acquired from experience and then appropriately applied. This learned knowledge is organized around concepts and *instances* (specific examples) of them.

Even for learning programs, some problems require search. Often the search space offers many alternatives, each of which may consume costly resources without adequate return. In such situations, the traditional AI response has been to plan, i.e., to reason about possible actions and their outcomes before committing to them. A *plan* here is an unexecuted set of steps expected to realize some goal, and *to plan* is to compose such a set of steps. There is extensive AI research on planning and its role in the simulation of intelligence.

In the last five years, however, some computer scientists have suggested that planning is not only unnatural but also unnecessary. They cite eloquent biological arguments that most animals do not plan (Baerends, 1976; Tinbergen, 1951). The ants that transport food cooperatively and the young birds that avoid precipices are hard-wired for their achievements. What is construed as intelligent behavior, say these computer scientists, can be simulated by purely *reactive systems* without any goals at all, ones that sense their environment and respond to it via effectors.

Brooks claims that it is possible to simulate human intelligence simply by response to sensory data, without explicit concept representation (1991). He provides the following "representation-free" criteria for a reactive system, confirming the earlier argument about implicit concept representation:

> "• Low-level simple activities can instill the Creature with reactions to dangerous or important changes in its environment.…
>
> • By having multiple parallel activities, and by removing the idea of a central representation, there is less chance that any given change in the class of properties enjoyed by the world can cause total collapse of the system.…
>
> • Each layer of control can be thought of as having its own implicit purpose.…
>
> • The purpose of the Creature is implicit in its higher-level purposes, goals, or layers.… " [Brooks, 1991]

Reactive systems are built from small components called *agents*. Although the program's internal representation of an agent need not be an if-then rule, it may be helpful to think of one that way. The "if" segment tests input signals, and the "then" segment specifies the appropriate output signal.[4] The scale postulated for agents ranges from the tiny and simplistic, like the neuron models called *bions,* to the somewhat more elaborate and complex, like circuits of finite state machines (Ring, 1991; Brooks, 1991). Each agent has a simple task to accomplish, e.g., avoiding a predator or deciding where to place a foot when walking (Cobb and Grefenstette, 1991; Hsu and Simmons, 1991). The agent "decides" what to do by processing input sensory data. The agent's reaction is its output. Such systems, with their small and specialized components, are very like the Society of Mind that Minsky has postulated (1986). The entire program performs as a collection of competing behaviors to which an observer may impute motives and goals where none are ever explicitly represented, i.e., reactive systems *deliberate* (i.e., plan from concepts) no more or less than ants do. The simulation of intelligence in reactive systems is therefore a control issue, without any concern for representation or focus of attention. If one simply provides accurate sensory input and the proper control mechanisms, the program will perform intelligently.

Proper control for a reactive system, of course, is non-trivial. Experiments with robots indicate that, even for simple tasks, a hierarchical *subsumption architecture* is the key to successful performance (Brooks, 1991; Connell, 1990). In such an architecture a task is decomposed into agent-sized components, and the agents are then implemented and incrementally assembled, layer by layer, into a functioning whole. A *layer* is a subsystem that produces an activity, i.e., pursues some purpose. Brooks, for example, has constructed a robot that explores by layering upon the ability to wander, which in turn is layered upon the ability to avoid obstacles. The agents' coordination is built-in and automatic. Such carefully engineered control systems require both human knowledge about the current task and human involvement in the debugging procedure as each layer is constructed.

Although building hard-wired control for a reactive system to simulate an ant, or even a young bird, is imaginable, constructing one for a two-year old human would be a monumental task. Developmental psychology suggests, moreover, that people do not come fully hard-wired, that they must *learn* much of their control systems. If people have preassembled low-level agents, the paucity, redundancy, and flexibility of human mental hardware suggests that the combination of those components to perform more elaborate tasks is learned, not innate (Blakeslee, 1991). Some reactive systems have already learned their own control strategy. Robots with a subsumption architecture have successfully learned to coordinate six individual leg-controllers to walk like an insect, and to coordinate lower-level behaviors to push boxes against a wall (Maes, 1990; Mahadevan and Connell, 1991). For reactive systems, as for other AI approaches, learning appropriate knowledge has become the key to successful specialization.

The question now becomes whether the appropriate knowledge to be learned is about explicit concepts, as in the traditional AI paradigms, or about control for a reactive system that operates without explicit representations. In a recent issue of *Artificial Intelligence*, Brooks and Kirsh took strongly opposing views on this matter (Brooks, 1991; Kirsh, 1991). Brooks argued that his preliminary successes with robots would scale up to any task because “there need be no explicit representation of either the world or the intentions of the system to generate intelligent behaviors for a Creature.” Kirsh, on the other hand, claimed that Brooks had worked only on *situation-determined behavior*, i.e., problems where an egocentric perception of the “indicators that matter” is sufficient to determine the appropriate course of action. Kirsh goes on to characterize *cerebral tasks,* the kinds of tasks on which he believes a reactive system would fail: tasks that involve other independent agents, that require planning, that require an objective viewpoint, that require problem solving.

The remainder of this paper examines how the cerebral task of game playing might be learned by a reactive system. The task is not to *solve* a game in the classical mathematical sense of deducing the best move for every situation, but to play the game as if the solution were known. Although the games referenced here are not as difficult as chess or checkers, they suffice to demonstrate the surprising challenges and possibilities for a reactive learning program, even in a simple task. Memory size becomes an important issue, one eventually solved by the incorporation of predefined explicit concepts.

2. Preliminary Reactive Playing

This section describes four ostensibly reactive programs that learn to play simple games reasonably well. Careful examination reveals their implicit concepts and their surprisingly large memory requirements. None of these programs learns as quickly as the average person, and only two of them learn to play tic-tac-toe[5] perfectly.

AI has traditionally approached game playing as a brute force task. The most successful chess programs, for example, are those that search the fastest and the furthest from the current *game state* (whose turn it is and how the playing pieces are placed on the board), and store the largest *opening book* of traditional contest beginnings (Berliner and Ebeling, 1989; Anantharaman et al., 1990).[6] These programs might be characterized as "search and more search, knowledge and more knowledge;" few of them learn. There have been, however, several recent implementations that model game playing purely as a learned response to pattern recognition, i.e., as a reactive system that learns to respond to sensory data that describes the game state.

An obvious reactive system for a specific game would construct one agent for each possible game state. Such an agent would output the perfect move whenever it sensed a match with its state description. Even so simple a game as tic-tac-toe has 5748 possible game states, however, and chess is estimated to have 10^{40}, so this approach is not destined for success. The agents are supposed to determine behavior, of course, but there ought to be a more memory-efficient and elegant construction for them. Each of the four programs described here senses[7] and reacts to recognized patterns without deliberating upon their meaning, i.e., their conceptual structure (Wierzbicka, 1985). The programs are introduced in increasing order of the amount of knowledge about game playing that they somehow incorporate into their control rules and/or their learning algorithms.

Painter has constructed a pattern-oriented program, Henri, that learns to play several different games on a three-by-three board (1992).[8] Henri learns the value of a sequence of three symbols (X's, O's, and blanks) and applies it to each of the eight possible three-position lines on the board. Pattern values are calculated by a primitive kind of reinforcement learning that uses the outcome of a contest to evaluate the patterns that occur in it. Henri learns, for example, that in tic-tac-toe "X-X-blank" is a better pattern for a line than "blank-X-blank." On its turn, Henri evaluates the patterns that would arise on the board if it were to make each of the possible legal moves, and selects the move that induces the highest pattern score. First, Henri watches while a programmed expert, which offers a broad variety of high-quality play, competes against itself in 50 contests. Then Henri plays 200 tic-tac-toe contests of its own against the programmed expert. Under this scenario Henri learns to play very well, but still imperfectly. It draws (the best anyone should be able to do against the expert) only about 85% of the time. Henri's failures come from inaccuracies in the pattern values. It is unclear how long Henri would take to play perfect tic-tac-toe, or if it ever would. *Pure pattern recognition seems insufficient for learning even this simple game in a noise-free environment.*

Gelfand has had similar results with N-N/Tree, a program that uses temporal differences and a neural net to learn values for nine-position patterns that describe games on a three-by-three board (Flax et al., 1990). N-N/Tree is also permitted a 3-ply search. It plays against a programmed expert that may err as often as 5% of the time. After 1000 tic-tac-toe training contests, N-N/Tree still loses 8% of its contests. Most intelligent people would learn to play such a simple game perfectly after that much experience; N-N/Tree does not. *Pattern recognition plus search can fail to learn this simple game in a realistic environment.*

Esfahany's classifier system, Dooze,[9] also learns to play games on a three-by-three board (Esfahany, 1992). Like N-N/Tree, a Dooze pattern lists all nine positions on a three-by-three board. Each classifier is a rule of the form "when position i is empty and the others have the following content, move to position i." The content of each position is described as an X, an O, a blank, or a "don't care" symbol. Dooze has several learning algorithms that introduce new classifiers into the system and delete others at the end of each contest. After 63 contests, on average, Dooze learns to play tic-tac-toe apparently perfectly by learning control rules, i.e., classifiers. Dooze's better performance may be attributable, however, to its larger memory requirements. There are $9 \cdot 4^8$ possible Dooze classifiers for tic-tac-toe. The program must maintain a set of at least 150 of them, about 15%, to learn to play expertly. Many of the learned classifiers are quite restrictive, i.e., entail patterns that would apply to very few game states. For five men's morris, a relatively simple game with 10 markers and 16 positions, 15% of the possible classifiers would be about 2^{31} rules. Some exploratory work with Dooze on games with slightly larger boards confirm that they are indeed likely to be a problem for this approach. *Learning only control, while adequate, may require more memory than a machine can offer.*

Levinson and Snyder have constructed Morph, a program that learns patterns to play chess (Levinson and Snyder, 1991). They describe Morph as a search-free and purely syntactic game player, i.e., one that reacts only to patterns, without planning or reasoning. A pattern in their system is a labeled graph that describes how selected markers and positions on the board relate to each other. Such a pattern is more sophisticated and less specific than the ones used by Dooze and N-N/Tree, and often applicable to more game states. Levinson has indicated that Morph's methods can be applied to any game once an appropriate pattern language is constructed.[10] Morph's approach was tested on tic-tac-toe; the program learned to play perfectly after approximately 250 contests and retained approximately 50 patterns to do so (Levinson, 1991). The difference in learning rate and storage requirements between this program and Dooze highlights *a possible learning tradeoff between memory size and number of training experiences required to learn.*

Careful analysis reveals that each of these "representation-free" programs actually incorporates concepts, generalizations about game playing understood by the programmer and incorporated into the code. Henri, the most visually oriented of them, only uses knowledge about lines and how positions may lie on them in two-dimensional space; its performance is also the weakest. N-N/Tree uses knowledge about the minimax algorithm for search control and how to apply it three-ply deep (Nilsson, 1980). Dooze's learning algorithms value a winning move (the last in a non-draw contest) highly, value every move the expert model makes, and recognize that what wins for X on one board wins for O on a board in which all the original X's are replaced by O's and the O's by X's. These constitute a hefty dose of primitive game-playing commonsense. By permitting a pattern-element to be a "don't-care symbol," Dooze also constructs abstractions, such as "If X holds the center, …." Morph's "syntactic" pattern language, when carefully examined, actually embeds ideas like threat and defense in both the pattern learner and in memory.

The four reactive game players have been shown here to employ concepts implicitly. Their memory requirements grow dramatically with the number of positions on the board, which makes their approach unlikely to scale up. Specialization for power can indeed be learned, as Morph shows, but more power seems to be related to a greater reliance on human knowledge of game-specific concepts. In particular, pattern generalizations are tailored to a single set of game-specific (Morph, N-N/Tree) or board-specific (Dooze, Henri) algorithms.

3. What is There to Learn?

Although concepts have been shown to impart power to reactive systems, the nature of such concepts remains to be clarified. This section describes four kinds of concepts, ways that people generalize about regularities in their experience: compiled knowledge, categories, scripts, and meta-principles. Cultural psychology is cited here to justify such concepts as the framework for human expertise in any domain.

People find it convenient to expect regularity in the world, and they have many devices to represent the regularities they detect. Each of the following is a representation of such experience.

• Some regular knowledge, like how to ride a bicycle, people *compile*, that is, abbreviate into a reliable response to specific situations. Compiled knowledge is experienced as reactive behavior; it has lost the detail, rationale, and instructions that once accompanied it. When the lost information is needed, reconstruction often requires observation from new experience. For example, one may know how to ride a bicycle without deliberation, but teaching another person to ride usually requires self-observation on some fresh bicycle-riding experience.

• People store some regularities as *categories*, sets of objects with common features (Wierzbicka, 1985). For example, people learn about chairs and that a chair usually has a back which supports weight. Every chair, physical or hypothetical, is an instance of the category, with specifically noted values for some of its features. When people sit in a chair, they assume that it has a back. The occasional unhappy surprise when this assumption fails testifies to the existence of the category and its associated expectations.

• Regularities about what is expected of an experience and those who participate in it are recorded as *scripts* (Schank and Abelson, 1977). For example, people learn that there is a partially ordered set of expectations for everyone's behavior in a restaurant. A visit to any restaurant is a walk through that script. Jokes about waiters may be funny because they violate that script.

• There are also regularities applicable to many different kinds of experience, ones to fall back upon when more detailed knowledge fails. For want of a well-established technical term, they are called *meta-principles* here. Examples of meta-principles include efficiency, safety, and propriety. The instantiation of a meta-principle for a particular domain results in a *principle,* behavioral guidance that may curtail search. The application of efficiency, safety, and propriety to driving a car, for example, would result in directives to drive rapidly, to drive carefully, and to obey the driving laws, respectively. Note the evident conflict among these principles.

People behave appropriately and learn quickly in part because they retrieve and apply these regularities, or concepts, continually and effectively. A single concept may be applied in a variety of problem-appropriate ways. For example, the chair concept, with its expectation of a back, can provide both search control for sitting and focus visual attention while furniture shopping. There is ample evidence that concepts are both learned and culturally determined, and that people prefer them to logical reasoning for any but the simplest examples (D'Andrade, 1991). In any culture, those judged experts are those who give more *modal responses*, i.e., agree most with commonly held regularities (D'Andrade, 1990). Thus an expert learns compiled knowledge, categories, scripts, and principles, and knows when and how to apply them.[11] Given those regularities, learning and problem solving with them may not be trivial, but it should be easier and require less memory.

The philosophy about general problem solving professed here is founded upon the preceding observations. An intelligent program, or person, retains a minimal amount of *useful knowledge* (knowledge that is expected to be relevant and may be correct) and then applies it in a flexible manner. *The regularities people learn, prefer, and exploit should be explicitly represented in an intelligent machine, along with the ways people use them.* These concepts include an appropriate behavioral script, category representations for problems and for useful knowledge that may be acquired during their solution, and principles for operating as an expert. The ways to apply them include how to learn useful knowledge and how to resolve conflicts that arise among principles, as they did in the driving example. A computer may be endowed with all this knowledge in advance, it may learn through trial and error, and it may learn by watching people solve problems. Just as animals have parents to imitate, an intelligent machine should have an intelligent model to observe. Given this characterization of concepts and their significance to expertise, it is now appropriate to construct systems that can capitalize on them.

4. A Program with Explicit Concepts Learns to Play

A more accessible research goal than the development of a general intelligence is the development of an *expert,* an intelligent learner for a broad, challenging set of related problems. This section describes an artifact in that direction: Hoyle, a program that learns to play two-person, perfect information board games.

Hoyle is based upon FORR, a general architecture for a learning and problem solving expert, one that postulates and capitalizes upon regularities (Epstein, 1991). As a result, Hoyle explicitly identifies different kinds of concepts, learns instances of them, and capitalizes upon those instances. As explained below, each of the four kinds of concepts appears in Hoyle: game and useful knowledge are categories, the game-playing algorithm is a script, the Learner is compiled knowledge, and the Advisors are instantiations of meta-principles. The discussion here describes these conceptual components and then explains how they are linked together to solve problems and to learn while playing against an external intelligent expert.

Hoyle's domain is the pre-defined category of two-person, perfect information board games. Each *game* Hoyle plays is an instantiation, a pre-specified, input instance, of that class. General knowledge about games is identified in advance by the programmer and coded into the system as slot names for the *game frame*. Table 1 is Hoyle's definition for tic-tac-toe. The slot names, somewhat revised here for clarity, are the game category definition, part of the general conceptual knowledge Hoyle begins with about all games. The slot values instantiate the frame for a particular game; they are the only specific knowledge Hoyle has about a new game before playing it. The function names in a game definition are references to Lisp code. Including the graphics routines to display the game on the screen, these functions are very brief, typically a total of less than 100 lines of code per game.

Name: *tic-tac-toe*
Marker for Player: *X*
Marker for Opponent: *O*
Initial-board: *(NIL NIL NIL NIL NIL NIL NIL NIL NIL)*
Scroll constant for screed display: *3*
Sliding move adjacencies: *nil*
Winning lines: *first row, second row, …, minor diagonal*
Dimensionality of board: *2*

Directions for user: *directions-tic-tac-toe*
Move input reader: *reader-tic-tac-toe*
Move filter: *legalp-tic-tac-toe*

Graphics display function for current state: *display-tic-tac-toe*
Move effector: *effector-tic-tac-toe*
Legal move generator: *generator-tic-tac-toe*

Predicate to detect end of contest: *endp-tic-tac-toe*
Predicate to calculate winner: *winp-tic-tac-toe*
Predicate to calculate loser: *lossp-tic-tac-toe*

Transformation from list to visual representation: *visualize-tic-tac-toe*
Transformation from visual representation to list: *devisualize-tic-tac-toe*

Table 1. The instantiated game frame for tic-tac-toe. Eight constants and eleven simple functions completely define the game.

Hoyle's *game-playing algorithm* is a script. It provides pre-defined, uniform, procedural direction to the program, so that it performs as if it were experienced in game playing, but not expert at any particular game. This script makes the program look like a game player: it detects when it is the program's turn to move, ensures that the participants alternately make legal moves, and announces the end of each contest, along with any winner. The game-playing algorithm also triggers the Learner when appropriate. Given a valid game definition and this script, Hoyle simulates a rule-abiding novice, one that makes legal, if not astute, moves.

Hoyle is given, in advance, both the right questions about its domain and heuristic procedures that may produce good answers to them. *Useful knowledge* is a category of questions about games; it is identified in advance by the programmer and coded into the system. Each question is represented as a slot name in the useful knowledge frame. Hoyle computes and stores one set of useful knowledge for each game. For example, a good question about a game is "Is it an advantage to go first?" Each of Hoyle's games has two *roles:* the participant who makes the first move is called *Player*, and the other is called *Opponent*. Each role is associated with a particular set of markers that appear on the board. If the question is interpreted as "Can Player always win with proper move selection?" then an affirmative answer requires a rigorous proof within the search space. If it is interpreted as "Does an expert typically win as Player but not as Opponent against another expert?" then an affirmative answer requires analysis of how a non-Hoyle expert's success relates to its role.

The *Learner* is compiled knowledge about how to answer the questions. For each useful knowledge slot the Learner has a uniform, heuristic, game-independent procedure to compute and store game-dependent slot values from experience at that game. The Learner's algorithms are identified in advance by the programmer and coded into the system. If the Learner were to retain everything it experiences, useful knowledge for an interesting game could quickly become unmanageably large. Hoyle, therefore, has learning algorithms that generalize and are highly selective about what they retain. Table 2 shows how those elements of Hoyle's useful knowledge applicable to this work might look like for tic-tac-toe after some learning. Further details on these slots appear in Section 6.

Returning to the example about whether Player has an advantage, the Learner knows that if every legal move from a game state results in a certain win for Player, then the game state itself is a certain win for Player. The program applies this, however, only to selected situations it has actually encountered in play, and allocates limited time to such consideration. Meanwhile the program also constructs a heuristic approximation to the answer, based upon the external expert's success in the role of Player. Recognition of an intrinsic advantage is, of course, different from knowing how to exploit one. To apply learned knowledge, Hoyle has Advisors.

Average contest length: *9*
Expected role winner: *draw*
Relevant forks: *none*

Openings: *the center is a draw, the corner is a win or a draw*
Significant states: *state-1, state-2, …, state-7*
Histories: *contest-1, contest-2, contest-3,*
Expert moves: *move-and-state-1, move-and-state-2, move-and-state-3*
Valid two-dimensional symmetries: *all*

Relevant Tier 3 Advisors: *all*
Most significant Tier 3 Advisors: *Candide, Worried, Greedy*
Voting methods: *all*

Table 2. An instantiated useful knowledge frame for tic-tac-toe. The slot names are known in advance; the slot values are learned.

An *Advisor* is a principle, implemented as a heuristic agent; its task is to make comments about choices when it is the program's turn to move. A *comment* is the Advisor's name, a legal move, and a *weight*, an integer from 0 to 10, indicating an opinion somewhere in the spectrum from strong aversion (0) to enthusiastic support (10). Each Advisor reacts to the fact that it is Hoyle's turn based upon the useful knowledge for the current game, i.e., *each Advisor is gradually specialized by experience*. In advance the programmer must recognize the cultural meta-principles, decide how to instantiate them as game-independent Advisors for the domain, determine the relevant useful knowledge for each Advisor, and code fixed weights for each situation in which an Advisor might choose to comment. The meta-principle of certainty, for example, prefers knowledge to risk. One way to instantiate it for Hoyle is as *Victory*, an agent that compares some segment of useful knowledge with the legal moves, and recommends with a weight of 10 each legal move that results in an immediate win. When it is Hoyle's turn, each Advisor may make any number of comments. Thus Victory could detect, and recommend, more than one winning move. There is no expectation that an Advisor is correct; it is a heuristic procedure. The origin of these Advisors is discussed in the next section, and the full complement of Hoyle's current Advisors is described briefly in the Appendix.

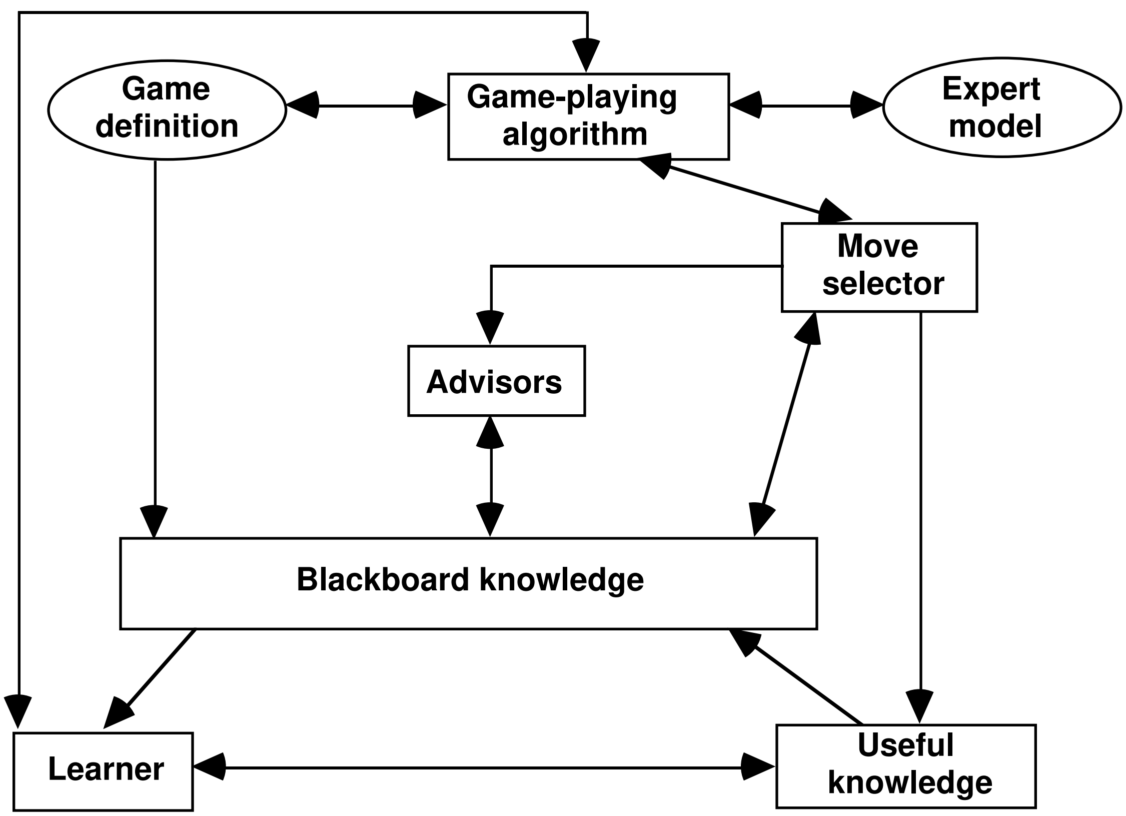

Figure 1. A schematic for Hoyle.

Hoyle's remaining components link its conceptual knowledge into the functioning program whose simple schematic appears in Figure 1. The arrows in the figure indicate the direction of information flow. The *expert model* is Kirsh's "other agent;" it may be either human or programmed. The expert model provides a behavioral paradigm for skill in the problem domain; it is only observed, never queried. Hoyle's expert model is either a very good human player or a hand-crafted program that offers a broad variety of independent, tournament-level competition. (See (Epstein, 1992) for additional details.) *Blackboard knowledge* is a temporary repository for remembered, sensed, and calculated information. Information flow on the blackboard is monitored by the *move selector*, which produces the final move decision. Given the definition of a game, the Learner initiates a tournament begun, managed, and terminated by the game-playing algorithm. Whenever it is Hoyle's turn to move, the game-playing algorithm places on the blackboard the current game state, the legal moves, and any useful knowledge about the game already acquired. (If Hoyle has had little or no experience at this particular game, there may be no useful knowledge.) The Advisors then calculate their comments and add them to the blackboard. Finally, a simple arithmetic calculation from the move selector identifies a move that is then forwarded to the game-playing algorithm for execution. After contests and tournaments, the Learner's algorithms compute and record useful knowledge for that game.

Hoyle benefits from explicit concept representation. It uses concepts to induce uniformity among games and to provide an organized focus of attention (useful knowledge). Although the program applies general learning methods (the Learner's algorithms), they are pre-specialized for game-playing and they learn values (for the useful knowledge slots) that appropriately specialize the Advisors' reactive behavior for each problem game. Unlike SOAR, which learns chunks, and PRODIGY which learns explanations, Hoyle learns concepts, domain-specific knowledge that provide power.[12] The identity of a learned concept is prespecified as a useful knowledge slot name, the procedure to learn it is prespecified as an algorithm to be executed on non-deterministically encountered data, and the application of learned knowledge is prespecified in one or many Advisors. How to recognize and apply useful knowledge is hard-wired, but the useful knowledge itself is learned. Hoyle begins as a program that reacts to situations, and then learns how to react to situations more expertly.

5. Why Hoyle is Reactive

This section argues that Hoyle meets Brooks' criteria for reactive systems. The discussion is supported with additional details on the Advisors and move selection. It begins with the origin of Advisors, described in the previous section as agents.

The degree to which Hoyle is reactive is the degree to which its components react rather than deliberate. In a robot domain, processes like "avoid" and "turn" compute physical effects; in a cerebral task, processes compute theoretical effects. Both are decisions about what to do based on what is sensed. Recall that an agent is supposed to decide what to do by processing input sensory data. To construct low-level agents, one must first address the question "How low is low enough?" Reactive systems have been explored for extremely low-level tasks, and have succeeded in learning appropriate "next-level-up" behaviors from them. Ring's program has incrementally learned more complex behaviors for movement hierarchies from single-behavior bions. Pierce's critter learns "how to use its sensorimotor apparatus and then how to navigate in its [simulated] environment" (Pierce, 1991). Mahadevan and Connell's robot learns to push boxes from the lower level behaviors for unwedge, push, and find. Given these successes, and the proposed cerebral task domain, it seems reasonable to assume that the agents need not address sensorimotor tasks. In Hoyle, for example, one can comfortably assume the ability to perceive markers on a game board, to move them, and to know whose turn it is. Hoyle simply puts its "sensory data" on the blackboard.

The origin of agents is presumed here to be the instantiation of the meta-principles operative in the culture (what D'Andrade calls the *cultural idea system*) for the specific task domain. Assume, for example, that game playing lies within some culture whose meta-principles include certainty, propriety, self-preservation, and deviousness. The instantiation of these meta-principles for the game-playing domain results in principles to be implemented as agents. Each agent in some way pursues the intent of its meta-principle.

There is often more than one way to instantiate a given meta-principle, and so there are likely to be many agents. The meta-principle of certainty, for example, prefers knowledge to risk. One way to instantiate it for the game-playing domain is as *Wiser*, an agent that compares some segment of useful knowledge with the current state and claims a contest win if there is a match. Another way to instantiate certainty is as *Sadder*, an agent that compares another segment of useful knowledge with the current state and resigns if there is a match. The meta-principle of certainty could also be instantiated by *Victory*, an agent that explores the legal moves and outputs any that result in an immediate win. The meta-principle of propriety would be instantiated by vetoing all but legal moves. The meta-principle of self-preservation could similarly give rise to *Don't Lose*, an agent that explores the legal moves and outputs a signal that erases from the blackboard any move that results in an immediate loss. Self-preservation can also give rise to *Panic,* an agent to block an immediate threat from the other participant to win in one move. Deviousness can produce *Enough Rope*, an agent that avoids blocking potential immediate losing moves for the other participant. Whatever its origin, *it is important to see each Advisor as merely an agent, a low-level intelligence that does not plan, that merely senses the blackboard data and responds to it with output signals.*

These output signals serve in turn as input to the decision mechanism, a kind of hierarchical subsumption architecture. Hoyle partitions its Advisors into a hierarchy of three tiers, each with its own internal conflict resolution technique. If any tier produces a decision, then none of the subsequent tiers is consulted. Control for the first two tiers is prespecified by the programmer. It represents that some Advisors are absolutely reliable and have a natural ordering based on what people know about problem solving.[13] In the first tier Wiser, Sadder, Victory, and Don't Lose are consulted in that order; if any of them has a comment to make, the others are never consulted. This is a representation that remembering is better than searching, and that winning quickly is primary. For the third tier, where the more heuristic Advisors lie, a simple voting procedure tallies all the comment strengths for each legal move and selects the one with the highest score. Ties are broken by random selection. The third tier simulates a set of parallel agents with finite resource allocations. Hoyle chooses a move when its Advisors react, one tier at a time, to the data on the blackboard. Regardless of the tier in which the output is computed, *it is important to recognize the move selection process as a rapid and simplistic mathematical computation, a reaction without search or deliberation.*

Hoyle meets the criteria cited in Section 1 for a reactive system as follows:

• "Low-level simple activities can instill the Creature with reactions to dangerous or important changes in its environment." The first two tiers contain quick reactions to short-term possibilities of success and failure.

• "By having multiple parallel activities, and by removing the idea of a central representation, there is less chance that any given change in the class of properties enjoyed by the world can cause total collapse of the system." Each Advisor processes sensory data independently. A change in Hoyle's world is a new game to play or new opposition to play against. The program is remarkably robust in the face of such changes, degrades gracefully, and has learned faster and better than was ever anticipated.

• "Each layer of control can be thought of as having its own implicit purpose." The purpose of an Advisor is to forward, in its own particular way, the meta-principle it instantiates. Hoyle is completely modular, i.e., it can run on any subset of its Advisors. Although variations in its ability to learn can result from the elimination of certain Advisors, it continues to function, i.e., plays legally.

• "The purpose of the Creature is implicit in its higher-level purposes, goals, or layers." There is no explicit representation, in Hoyle's game-playing algorithm or in its control mechanism, of intention, belief, plan, goal, subgoal, win, loss, or draw. The execution cycle is pause-sense-react, where “pause” cedes control to the expert model. Hoyle’s game-playing algorithm referees each contest, enforces the rules, notifies the other participant when it is time to move, and triggers the appropriate learning agents when a contest ends. “Sense” is the collection of the blackboard information, and “react” is the collective response of the agents to the input on the blackboard.

Thus Hoyle is a reactive system. It does not plan; its computations are cerebral versions of mechanical sensory devices.

6. The Power of Concepts and Memory

This section examines Hoyle’s ability to learn to play twelve games expertly. Although Hoyle is reactive, the full version of the program incorporates and remembers concepts, knowledge about the regularities that people learn, prefer, and exploit when playing games, and how people use those regularities. When the program is partially disabled and the results observed, it is clear that the memory and application of those concepts (the compiled knowledge, categories, script, and principles) provide the program with its power.

A *performance curve* for a tournament plots the cumulative number of contests where the program achieved the result an expert would have, against the cumulative number of contests it played. Figure 2 shows performance curves for several 20-contest tic-tac-toe tournaments with various configurations of the program. The top line in Figure 2, for absolute expertise, shows what happens when one perfect player competes in tic-tac-toe against another. When Hoyle played at random (curve #1 in Figure 2), making arbitrary legal moves against a programmed expert, it lost all but one contest, a lower bound on performance. Once Hoyle learns to play perfectly, its curve should parallel that for absolute expertise. The gap between the curve for absolute expertise and curve #1 indicates the extent of the learning task.

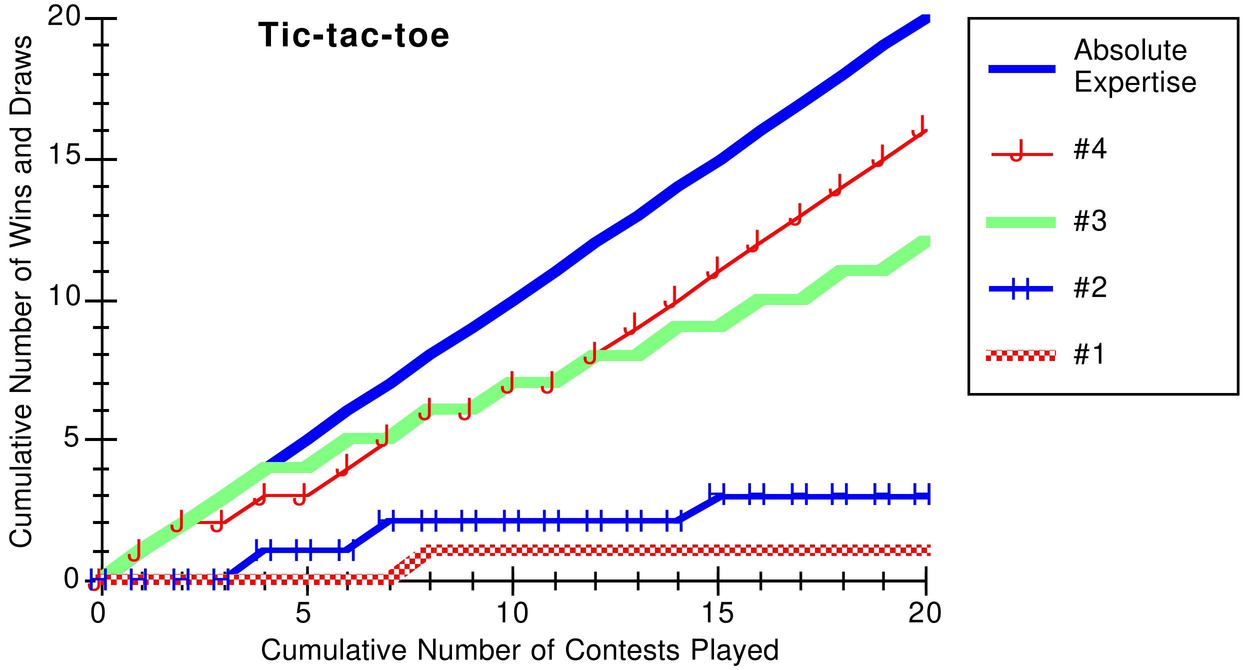

Figure 2. Cumulative non-losses in four tic-tac-toe tournaments of twenty contests each.

Hoyle's useful knowledge is a compendium of the regularities expert game players look for and exploit. One element of useful knowledge is the *significant state*, a state that will inevitably be won or lost when both participants play expertly. Such a state is a deduced, compiled regularity computed by the Learner at the end of a contest and stored in memory. A significant win state is stored with any known, legal next move guaranteed to lead to a win;[14] a significant loss state is recorded only as a certain loss. The regularity captured by a significant state is that *every* time it occurs the outcome when two experts play is inevitable, not that it is an abstraction of a game state. A significant state may be either be treated as a concept (used in computation) or treated as a reflex action (turned toward or avoided). Retrieval of significant states is from a hash table, and is assumed to require no search. Besides significant states, as Table 2 shows, useful knowledge includes selected contest histories, selected openings, moves experts have made that may have served them well, whether or not it is an advantage to go first, the length of an average contest, applicable two-dimensional symmetries, statistical observations on the relevance and trustworthiness of individual Advisors, and relevant *forks*.[15] Most of the useful knowledge requires only a fixed amount of space regardless of the difficulty of the game. Empirically, the memory requirement for learning a game is essentially a function of the number of significant states and expert moves.

At present Hoyle plays correctly (has the rules for) 24 games gathered from almost as many cultures. Although none of Hoyle's games is as difficult as checkers or chess, they incorporate a variety of challenges: stages where the rules change, cycles, and very large search spaces. One of the games it learns to play expertly is Qubic, a three-dimensional version of tic-tac-toe on four parallel four-by-four grids. Qubic has more than a billion game states and is generally acknowledged to lie on the border between simple games and the difficult ones.

Hoyle learns against an expert model, but human experts tire, and for many of these games there are no human experts available. Therefore, Hoyle usually learns against a hand-crafted model program, one that offers a broad variety of high-quality play. Each model program represents a compendium of the best human knowledge about its game, and requires substantial design and debugging. At this writing, Hoyle has model programs for 12 of its games, and has learned to play all of them perfectly, none with more than 100 significant states. Tests of its prowess at the other 12 games for which it has the rules await information-rich learning environments, i.e., expert model programs.

The performance of the full version of Hoyle at tic-tac-toe against a programmed expert is shown as curve #4 in Figure 2. (Similar results occur against human challengers.) In 12 contests, Hoyle learns to play perfectly, and stores an average of nine significant states and expert moves to do so.[16] Compare this with Dooze's 150 rules and 63 contests, and the

Morph-like 50 patterns and 250 contests.

The Learner's algorithms and each Hoyle Advisor, as already observed, embody one or more concepts about game-playing. Elimination of all concepts from Hoyle would deprive it of its game-playing script, and make it unable to play at all. Less radically, if all the Advisors and all learning were removed, Hoyle would make random moves and would play no better than curve #1 in Figure 2. A more interesting reactive learning version of Hoyle, one with severe concept restrictions, would be analogous to a Brooks-style robot that gradually redid its own circuitry, i.e., learned responses to game states but could use them only in one way. This *concept-poor* version of Hoyle would be a flawless imitator; the Advisors would react to the blackboard knowledge but would not perform simple calculations from past experience. The program is thus restricted when its learned useful knowledge is limited to detailed recollection of the contests it has played and of significant states (as reflexes, not as concepts), and then it is permitted only performance repetition, rather than simple computation, with its knowledge. This concept-poor version recognizes previously encountered certain wins and losses, imitates moves the expert made in the identical situation, and tries to avoid reproducing its own failing moves in an identical situation. The performance of this concept-poor version against a programmed expert is shown as curve #2 in Figure 2. (Similar results occur against humans.) The partially-disabled Advisors immediately learn and recommend the successful openings of the human participant, but find the play later in a contest more difficult. In this tournament Hoyle very gradually learns to avoid losing moves in the tic-tac-toe endgame through significant states, and loads up its memory with them. In essence, Hoyle is now required to solve the game. Although theoretically Hoyle could learn about all possible states by backing up such experience, Figure 2 shows that it is a slow process.

Of course, Hoyle is not limited to tic-tac-toe. In another simple game, pong hau k'i, the concept-poor version of the program learned to play perfectly immediately. A few other games also appeared to be slowly learnable. Other reasonably simple games, however, were so difficult for the program that the observers did not have the patience to wait while it gradually accumulated a large library of appropriate responses by making what seemed to be every possible error. The reactive programs of Section 2 employed some basic concepts to reduce memory and improve learning speed, an approach from which a concept-poor Hoyle could also benefit.

A *memory-free* but concept-dependent version of Hoyle would be analogous to an intelligent participant who played every contest at the same game as if it were the first. Perhaps the Advisors that apply useful knowledge do not really need memory to learn to play expertly. For example, Pitchfork is Hoyle's Advisor on forks; it applies useful knowledge on relevant forks, if it has any. But how would Pitchfork, and the rest of the Advisors, do without memory? To answer this question, Hoyle played a tournament against a human expert (curve #3 in Figure 2), this time only with those Advisors omitted from the concept-poor version and without memory. Now Hoyle had to rely only on its concepts. In the fifth tic-tac-toe contest, the person, who had no knowledge of Hoyle's memory lack, tried a simple strategy for X that Hoyle failed to combat. The learning algorithm would normally learn to avoid that mistake the next time, but the memory-free version could not. The human repeated the same strategy at every subsequent opportunity, in each contest where Hoyle played O, and quickly observed that Hoyle did not learn from its mistakes. (This accounts for the step-like pattern of curve #3 in Figure 2; Hoyle played expertly in alternate contests, as X.) Thus the program without memory plays a reasonably intelligent tic-tac-toe contest, but performs unintelligently in a tournament situation. Learning requires memory, and without memory Hoyle never would develop expertise.

Table 3 summarizes the differences in memory, learning time, and the resultant quality of play for Hoyle and its memory-free and concept-poor versions. The full version of the program learns to play perfectly after an average of 12 contests, and requires an average of

9 additional storage positions in memory beyond the minimal allocation for useful knowledge.[17] The memory-free version can never learn to play perfectly against an observant human armed with the simple strategy described above; without storage it will lose every contest in which it is Opponent. The concept-poor version may, after many contests, eventually learn all the hundreds of significant states and expert moves that would hard-wire it for success. This will be a slow task, however, bounded from above only by the size of the search space, adjusted for symmetry. Michie's experiments with MENACE, his matchbox tic-tac-toe machine indicate that a set of 300-odd expert moves can be learned in about 150 contests to simulate perfect play (Michie, 1986). From his description it is estimated that Hoyle, after the same number of contests, would play as well when it learned approximately 300 expert moves and 100 significant states, for a total memory requirement of roughly 400 storage positions. Although the data in Table 3 is for tic-tac-toe, in all the games, except the very easiest, it is repeatedly observed that *Hoyle's power derives from this synergy between memory and concepts.*

	Hoyle	**Memory-free**	**Concept-poor**
Memory	9 units	none permitted	~ 400 units
Learning time	12 contests	cannot learn perfectly	150 contests
Quality of play	perfect	50% losses	perfect

Table 3. Memory, learning time, and the resultant quality of play for the full version of Hoyle, and for its memory-free and concept-poor versions.

AI researchers must be leery of claiming a result from experiments only with small problems; there are many cases where methods do not *scale up* well, i.e., reproduce their success on more difficult tasks. An obvious way to scale up is to have Hoyle play more difficult games. Because the intent of this research has been to study games that a variety of people have found challenging for hundreds of years, the games Hoyle learns come from two anthropology books on games in diverse cultures (Bell, 1969; Zaslavsky, 1982). Hoyle is essentially intended to work through Zaslavsky's book, with occasional insertions from Bell, in an order of difficulty intuited by several expert game players.

It is not always possible to predict in advance how difficult it is to become expert at a specific game. Experience with Hoyle has been that the size of the search space, i.e., the number of possible game states, while relevant, is not the determining factor. For example, the game of lose tic-tac-toe is identical to tic-tac-toe except that the object is to *avoid* placing three of one's markers in the same column, row, or diagonal; whoever does so, loses. The two games have isomorphic search spaces, albeit with different win and loss labels, but lose tic-tac-toe is a much more difficult game to learn, both for people and for Hoyle. Another possible metric for level of difficulty is the *branching factor,* the average number of legal moves from which to choose on any turn. For Hoyle's learning algorithms, games with the same size search space are more difficult to learn when their branching factors are larger, because it may take longer to explore all the alternatives. Another possible predictor for level of difficulty is the kinds of moves permitted by the rules. There are *placing moves*, where a marker appears on the board for the first time, *jumping moves,* where a marker is lifted from one position on the board and placed on another, and *sliding moves*, where a marker is shifted from one position to another on the board along a line unimpeded by other markers. Thus far there appears to be no significant distinction among games with any of these move types. A *capture move* is one where any of the previous move types results in a board configuration where the rules require that the mover then remove from the board one from a specified subset of the other participant's markers. Games that offer a choice of more than one marker to capture seem to be more difficult to learn, probably because they increase the branching factor.

Thus far Hoyle has learned to play perfectly, without planning and with only minimal search, every game for which it has an expert model program or expert human competition. When Hoyle has had difficulty learning a new game, its useful knowledge has been very gradually extended to include new concepts and the low-level agents to apply them. This gradual debugging process is much like Brooks' robot control layering: "so far, so good." There are long-range plans for Hoyle to consider analogies among games, but no substantial work on this yet.

Hoyle not only has concepts, it applies them flexibly, i.e., different Advisors apply the useful knowledge in a variety of ways. The average length of a contest, for example, is relevant to whether or not an opening may still be in progress, as well as to whether an endgame strategy is worth considering. Six different Advisors use the same significant state cache to construct comments. Looking ahead one or two moves, never further, is an important way that Hoyle applies useful knowledge.

Hoyle outperforms the four programs of Section 3, this paper has argued, because it explicitly represents relevant concepts. As a result it is able to learn with far smaller memory requirements and to apply this compact useful knowledge flexibly.

7. Conclusions

Before experiments with Hoyle began, its creators had only general ideas about the specific concepts Hoyle would need and the ways it would apply them. For example, although Advisors that make comments based on *mobility* (number of legal moves available) and *material* (number of markers each participant has on the board) were debugged and ready, Hoyle managed to learn its first twelve games without them. One of these readily learned games is, to all appearances, simply a matter of not becoming trapped, i.e., of mobility; the rules state that as long as you can move you have not lost. Hoyle learned to play this game expertly without the mobility Advisor; it just imitated the learned moves of the model program. The lesson here is that *more high-level behavior is available through low-level* *reactive processes than one might initially suspect.*

Eventually, however, the games become sufficiently difficult to require additional concepts. For example, preliminary results indicate that Hoyle will be unable to learn to play some games with capture moves expertly without the Advisor for material. The program simply sees no reason to prefer a state in which the other participant has fewer markers; it resorts once again to the tedious process of making all possible errors and the memory-greedy process of remembering them. Once the Advisor for material is included, Hoyle's play improves dramatically. The lesson here is that *learning high-level behavior efficiently with a limited memory requires concepts.* Hoyle's concepts organize the way it remembers experience (useful knowledge), focus its attention on what is important to learn (useful knowledge), force it to apply its experience (game-playing algorithm and Advisors), and permit it to discard experience that is judged unlikely to be useful (learning algorithms). This author suspects that the number of concepts required to play well, and the number of ways in which those concepts interrelate, is the true metric for game difficulty.

One recent surprise has been that, in a relatively simple game with only six markers and nine positions, the strategy postulated for the expert model program by preliminary analysis was far less effective than some perceptual patterns encountered by an undergraduate student who is a good chess player. Although the student eventually worked up a logical explanation in mathematical terms, it was clear from his psychological protocol that *first* he noticed patterns, and *later* he thought about why they had the observed effect. Unlike Morph, the student was drawn to purely visual patterns, ones that did not incorporate ideas about threat and defense.[18] The lesson here is that *some high-level behavior, though people may prefer to anthropomorphize it in conceptual terms, has its origins in reactive processes.*

Hoyle's ability to learn which of the eight symmetries of the two-dimensional plane are applicable to a game is a recent improvement. The program then applies those symmetries to minimize both memory requirements and search time. For example, Hoyle recognizes only three possible distinct opening moves for tic-tac-toe: the center, a corner, and a side square. Hoyle with symmetry learns faster and requires less memory, as one would expect. The speedup observed is roughly linear in the number of positions on the board. The lesson here is that *low-level sensory data can offer an immediate improvement in high-level processing.*

For the time being, several tasks have been relegated to the human system designer: the correct identification of the culturally determined meta-principles (characterized as "commonsense" but by no means trivial), the instantiation of the meta-principles to construct low-level agents, the assignment of Advisors to tiers based upon knowledge about relations among meta-principles, the specification of which agents access which concepts, and the description of how they apply that knowledge. This author believes that all of these can eventually be automated.[19]

Meanwhile work continues on the specified sequence of games. For the moment Hoyle manages to hold search to a minimum, and not to plan. Future research includes additional analysis of sensory data to learn patterns and then apply them, i.e., to become more reactive.

In a domain that is not situation-determined, Hoyle is a successful, reactive, hierarchical system that retains only a small fraction of what it experiences. The program pays an interesting price for its reactivity, however. It must rely on concepts to learn to perform intelligently. Hoyle offers evidence that learning cerebral tasks demands more concepts than Brooks would like, and far fewer than Kirsh would assume. Hoyle may not resolve the control-concept controversy, but it should certainly influence our attitude on the significance of low-level agents in high-level tasks.

Hoyle's results demonstrate for at least one broad cerebral task, game playing, that a reactive system without memory is impractical, and that reliance only on extensive, detailed memory is brittle and often impossible. This paper has shown how concepts can structure resource-efficient memory, provide flexibility, and regularize knowledge to support performance. What Hoyle has begun to realize is a reactive strategy that relies on memory for learning and on concepts for power. Will a reactive program ever, then, have to search and plan and believe? Hoyle's answer is not yet, perhaps not explicitly, and far less than might have ever been expected.

Acknowledgments

The author thanks Jack Gelfand, Alice Greenwood, Cullen Shaeffer, Rick Shweder, and two anonymous referees for their insightful comments and constructive suggestions. Michael Georgopoulos provided expert-level programming support.

Notes

1. Concepts arbitrarily created by some authority are learned by rote, and excluded here.

2. Edmond Hoyle was an eighteenth century expert on games. He codified the rules for many popular games of his day.

3. A *perfect information game* is one in which all participants have full access to all relevant information, e.g., no concealed hands as at poker. When skilled human game players explain what they believe their internal processes to be, such game playing is clearly a cerebral activity, fraught with planning and belief systems.

4. Brooks might add here that his processes have no variables and do no matching. Watching "to see if there is anything dead ahead," however, can be construed as a kind of matching with an implicit variable.

5. Tic-tac-toe, also known as naughts and crosses, is played on a three-by-three grid. The first participant has five X's, and the second has four O's. Initially the board is empty. A turn consists of placing one of your markers in any empty square. The first one to place

three of the same markers in a row, vertically, horizontally, or diagonally, wins. There are eight such winning lines. If no win is achieved, play ends in a draw when there are no more empty squares.

6. It is important to distinguish here among a game, a contest, and a tournament. A *game* is an activity with a board, playing pieces, and rules. A *contest* is an experience playing a game, beginning at an initial situation specified by the rules and ending when the rules declare a winner or a draw. A *tournament* is a sequence of contests at a specific game in which the participants alternately go first or second. For example, chess is a game at which two people might play a tournament of 16 contests.

7. For robots, sensation may entail a sonar device that outputs a version of reality. In a cerebral task, sensation is not necessarily a physical process.

8. It is important to distinguish here between a *position* (a location on the board where a piece may reside) and a *marker* (a playing piece). Thus in tic-tac-toe there are nine legal positions on which a participant may place at most one of the five markers for X and four markers for O.

9. Dooze is the Persian name for tic-tac-toe.

10. Since this language must be hand-crafted, Morph itself is not a system that can play more than one game.

11. Different regularities develop in different realms, e.g., compiled knowledge arises during repeated, often solitary, practice, while scripts are part of social life. There is no reason to presume that some uniform learning method is applicable to all of them.

12. Hoyle can also learn, as part of its useful knowledge, improvements to its default move selector. Some control is permanently hard-wired by nature of the Advisors and their response times, but some control (the last three slots of the useful knowledge frame in

Table 2) is refineable with experience, as if Hoyle could partially rewire itself. Control learning is beyond the scope of this paper, however. See (Epstein, 1990b) for further details.

13. This is a version of Brooks' *suppression* and *inhibition* mechanisms.

14. Note that there may be more than one optimal next move and each such move is stored as an appropriate reaction. What is *not* stored is plans, sequences of moves that would, when executed, achieve the win.

15. A *fork* is a game-independent meta-plan whose instantiation with the current game state requires costly matching, but can provide powerful offensive and defensive advice. Epstein gives a detailed explanation of forks, their application, and the explanation-based algorithm the Learner uses to learn which additional forks are important to a specific game (1990a).

16 . When in doubt, Hoyle may choose at random from some subset of the legal moves its Advisors find attractive. Thus the program structures its own non-deterministic learning environment, and averages are necessary to report its performance.

17. When in doubt, Hoyle may choose at random from some subset of the legal moves its Advisors find attractive. Thus the program structures its own non-deterministic learning environment, and averages are necessary to report its performance.

18. This is more closely related to the phenomenon of *chunking* (e.g., Campbell, 1988).

19. Admittedly, the allocation of the difficult tasks to the system designer in some way begs the philosophical question as to what an ultimate general intelligence might be. The more modest goal targeted here, as previously stated, is only the simulation of a skilled expert.

Appendix: The Advisors and Move Selection

Hoyle's Advisors provide advice on move selection when it is Hoyle's turn. Their implementation is game-independent. The Advisors are partitioned into tiers. If any tier produces a single move recommendation, then none of the subsequent tiers is consulted. The tiers, and their internal conflict resolution technique are summarized below. Because Hoyle plays a broad variety of games, certain Advisors may not be relevant to a specific game.

Tier 1:

Within this tier Advisors are consulted in the indicated order. If any of the first three Advisors makes one or more comments, the remainder of the Advisors is not consulted and control returns to the game-playing algorithm.

• *Wiser:* If this is a significant win state, make the correct move. This Advisor is completely dependent upon memory.

• *Sadder:* If this is a significant loss state, resign. This Advisor is completely dependent upon memory.

• *Victory:* If you see a winning move now, take it.

• *Don't Lose:* Do not make a move that will result in a loss. This Advisor has a memory component and may reduce the set of legal moves.

Tier 2:

Within this tier Advisors are consulted in the indicated order. If the first two Advisors make one or more comments, the remainder of the Advisors is not consulted and control returns to the game-playing algorithm.

• *Panic:* If the non-mover would have a winning move from this board, block it. If all such winning moves for the opposition are not blockable by a single move, resign.

• *Shortsight:* Advise for and against moves based on a two-ply lookahead. This Advisor has a memory component and may reduce the set of legal moves.

• *Enough Rope:* If the non-mover would have a losing move from this board, avoid blocking it, i.e., leave the non-mover the opportunity to err. This Advisor may reduce the set of legal moves.

Tier 3:

Within this tier all Advisors are consulted before control returns to the game-playing algorithm. Conflict resolution is based on the frequency and weight of the comments for and against each move.

• *Pitchfork:* Advance offensive forks and destroy defensive ones. This Advisor has a memory component.

• *Candide:* Formulate and advance naive offensive configurations, i.e., ones that do not anticipate any opposition.

• *Worried:* Observe and destroy naive offensive configurations for the other participant, i.e., ones that do not anticipate any opposition.

• *Open:* Recommend previously observed non-Hoyle openings. This Advisor is completely dependent upon memory.

• *Anthropomorph:* Move as a winning or drawing non-Hoyle expert once did. This Advisor is completely dependent upon memory.

• *Not Again:* Do not move as a losing Hoyle once did. This Advisor is completely dependent upon memory.

• *Cyber:* Move as a winning or drawing Hoyle once did. This Advisor is completely dependent upon memory.

• *Greedy:* Propose moves that advance more than one winning configuration.

References

Amarel, S. (1968), 'On Representations of Problems of Reasoning about Actions,' in D. Michie, ed., *Machine Intelligence 3*, Edinburgh: Edinburgh University Press, pp. 131-171.

Anantharaman, T., Campbell, M. S., and Hsu, F.-h. (1990), 'Singular Extensions: Adding Selectivity to Brute-Force Searching,' *Artificial Intelligence* **43**, pp. 99-110.

Baerends, G. P. (1976), 'The Functional Organization of Behavior,' *Animal Behavior* **24**, pp. 726-735.

Bell, R. C. (1969), *Board and Table Games from Many Civilizations*, London: Oxford University Press.

Berliner, H., and Ebeling, C. (1989), 'Pattern Knowledge and Search: The SUPREM Architecture,' *Artificial Intelligence* **38**, pp. 161-198.

Blakeslee, S. (1991), 'Brain Yields New Clues on its Organization for Language,' New York: *The New York Times,* September, 10, p.C1.

Brooks, R. A. (1991), 'Intelligence without Representation,' *Artificial Intelligence* **47**, pp. 139-160.

Campbell, M. S. (1988), 'Chunking as an Abstraction Mechanism,' Ph.D. Thesis, Carnegie Mellon.

Cobb, H. G., and Grefenstette, J. J. (1991), 'Learning the Persistence of Actions in Reactive Control Rules,' *Proceedings of the Eighth International Machine Learning Workshop*, pp. 293-297.

Connell, J. (1990), *Minimalist Mobile Robotics: A Colony-style Architecture for an Artificial Creature*, New York: Academic Press.

D'Andrade, R. G. (1990), 'Some Propositions about the Relations between Culture and Human Cognition,' in J. W. Stigler, R. A. Shweder and G. Herdt, ed., *Cultural Psychology: Essays on Comparative Human Development*, Cambridge: Cambridge

University Press, pp. 65-129.

D'Andrade, R. G. (1991), 'Culturally Based Reasoning,' in A. Gellatly and D. Rogers, ed., *Cognition and Social Worlds*, Oxford: Clarendon Press,

Epstein, S. L. (1990a), 'Learning Plans for Competitive Domains,' *Proceedings of the Seventh International Conference on Machine Learning*, pp. 190-197.

Epstein, S. L. (1990b), 'Learning to Control a Blackboard System for Game Playing,' *Proceedings of the AAAI Workshop on Blackboard Systems*.

Epstein, S. L. (1991), *Learning under a Weak Theory*, Department of Computer Science, Hunter College.

Epstein, S. L. (1992), 'Hard Questions about Easy Tasks - Issues from Learning to Play Games,' in *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*, Cambridge, MA: MIT Press. To appear.

Esfahany, K.H. (1992), in preparation.

Flax, M. G., Gelfand, J. J., Lane, S. H. and Handelman, D. A. (1990), Integrating Neural Network and Tree Search Approaches to Produce an Auto-Supervised System that Learns to Play Games. In *Proceedings of The Aerospace Applications of Artificial Intelligence Conference,*

Goodman, N. (1973), *Fact, Fiction, and Forecast*, Indianapolis, IN: Bobbs-Merrill.

Hsu, G.-T., and Simmons, R. (1991), 'Learning Footfall Evaluation for a Walking Robot,' *Proceedings of the Eighth International Machine Learning Workshop*, pp. 303-307.

Kirsh, D. (1991), 'Today, the Earwig, Tomorrow Man?,' *Artificial Intelligence* **47**, pp. 161-184.

Laird, J. E., Rosenbloom, P. S., and Newell, A. (1987), 'SOAR: An Architecture for General Intelligence,' *Artificial Intelligence* **33**, pp. 1-64.

Lenat, D. B. (1976), 'AM: An Artificial Intelligence Approach to Discovery in

Mathematics,' Ph.D. Thesis, Department of Computer Science, Stanford University.

Levinson, R., (1991), Personal communication.

Levinson, R., and Snyder, R. (1991), 'Adaptive Pattern-Oriented Chess,' *Proceedings of the Eighth International Machine Learning Workshop*, pp. 85-89.

Lindsay, R. K., Buchanan, B. G., Feigenbaum, E. A., and Lederberg, J. (1980), *Applications of Artificial Intelligence for Organic Chemistry: The Dendral Project*, New York: McGraw-Hill.

Maes, P., and Brooks, R. A. (1990), 'Learning to Coordinate Behaviors,' *Proceedings of the Eighth National Conference on Artificial Intelligence*, pp. 796-802.

Mahadevan, S., and Connell, J. (1991), 'Scaling Reinforcement Learning to Robotics by Exploiting the Subsumption Architecture,' *Proceedings of the Eighth International Machine Learning Workshop*, pp. 328-332.

Michie, D. 1986. *On Machine Intelligence.* Second edition. Ellis Horwood Ltd. **check**

Minsky, M. (1986), *The Society of Mind*, New York: Simon & Schuster.

Minton, S. (1988), *Learning Search Control Knowledge - An Explanation-Based Approach*, Boston: Kluwer Academic.

Mitchell, T., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M. N., and Schlimmer, J. C. (1990), 'Theo: A Framework for Self-Improving Systems,' in K. Vanlehn, ed., *Architectures for Intelligence*, Boston: Erlbaum,

Newell, A., Shaw, J. C., and Simon, H. A. (1963), 'Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics,' in E. A. Feigenbaum and J. Feldman, ed., *Computers and Thought*, New York: McGraw-Hill,

Nilsson, N. J. (1980), *Principles of Artificial Intelligence*, Palo Alto: Tioga Publishing.

Painter, J. (1992), 'Pattern Recognition for Decision Making in a Competitive Environment,' Master's Thesis, Department of Computer Science, Hunter College of the City University of New York.

Pierce, D. (1991), 'Learning a Set of Primitive Actions with an Uninterpreted Sensorimotor Apparatus,' *Proceedings of the Eighth International Machine Learning Workshop*, pp. 338-342.

Ring, M. (1991), 'Incremental Development of Complex Behaviors through Automatic Construction of Sensory-motor Hierarchies,' *Proceedings of the Eighth International Machine Learning Workshop*, pp. 343-347.

Sacerdoti, E. D. (1974), 'Planning in a Hierarchy of Abstraction Spaces,' *Artificial Intelligence* **5**, pp. 115-135.

Schank, R., and Abelson, R. (1977), *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*, Hillsdale, NJ: Erlbaum.

Shortliffe, E. H. (1976), *Computer-Based Medical Consultations: MYCIN*, New York: Elsevier.

Tinbergen, N. (1951), *The Study of Instinct*, Oxford: Oxford University Press.

Wierzbicka, A. (1985), *Lexicography and Conceptual Analysis*, Ann Arbor, MI: Karoma Publishers.

Zaslavsky, C. (1982), *Tic Tac Toe and Other Three-in-a-Row Games, from Ancient Egypt to the Modern Computer*, Crowell.