This article appeared in *The Encylopedia of Cognitive Science,* 2002 London, Macmillan Reference Ltd.

Game-playing Programs

Article definition: Game-playing programs rely on fast deep search and knowledge to defeat human champions. For more difficult games, simulation and machine learning have been employed, and human cognition is under consideration.

Game trees

Board games are not only entertaining — they also provide us with challenging, well-defined problems, and force us to confront fundamental issues in artificial intelligence: knowledge representation, search, learning, and planning. Computer game playing has thus far relied on fast, deep search and vast stores of knowledge. To date, some programs have defeated human champions, but other challenging games remain to be won.

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*). Examples appear in Figure 1. Each game has its own finite, static set of rules that specify legal locations on the board, and when and how contestants may *move* (transform one state into another). The rules also specify an *initial state* (the starting position for play), a set of *terminal states* where play must halt, and assign to each terminal state a *game-theoretic value*, which can be thought of as a numerical score for each contestant.

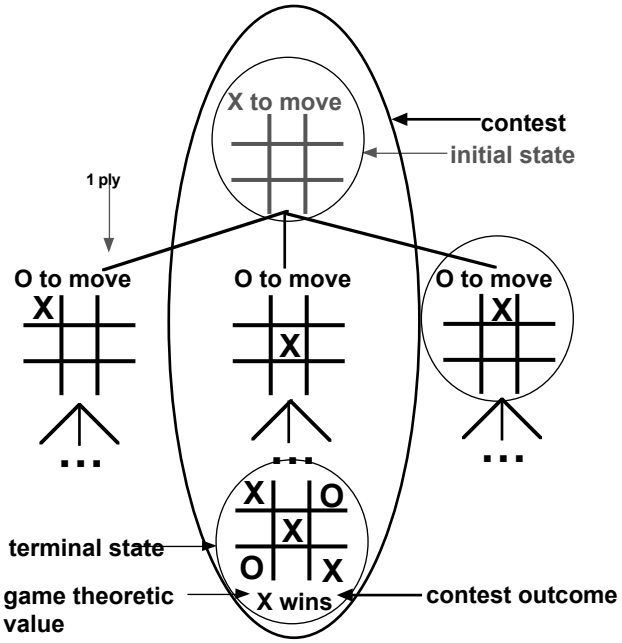


Figure 1: A game tree and basic game playing terminology.

As in Figure 1, 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. 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 goal of each contestant is to reach a terminal state that optimizes the game-theoretic value from its perspective. An *optimal 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 optimal. 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 stops at states all of which are labeled with values, a *minimax algorithm* backs those values up, one ply at a time, selecting the optimal move for the mover at each node. In Figure 2, for example, each possible next state in tic-tac-toe is shown with its game theoretic value; minimax selects the move on the left.

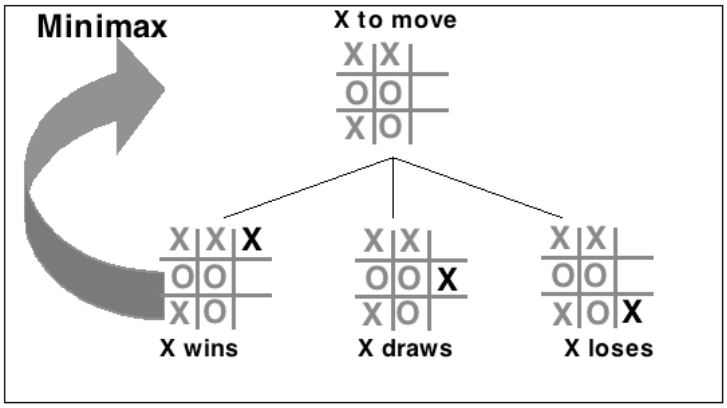


Figure 2: A minimax algorithm selects the best choice for the mover.

Retrograde analysis backs up the rule-determined values of all terminal nodes in a subtree to compute the game-theoretic value of the initial state. It minimaxes from all the terminal nodes to compute the game-theoretic value of every node in the game tree, as shown in Figure 3. 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 any challenging game, such as checkers (draughts) or chess, retrograde analysis to the initial state is computationally intractable. Therefore, move selection requires a way to compare alternatives.

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 game trees, one can generate a perfect evaluation function by caching the values from retrograde analysis along with the optimal 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 described as a set of *features*, descriptive properties such as piece advantage or control of the center. It is possible, for example, to construct, and then program, a perfect, feature-based evaluation function for tic-tac-toe. Given a

perfect evaluation function, a game-playing program searches only one ply — it evaluates all possible next states and makes the move to the next state with the highest value. For a challenging game, however, the identity of the features and their relative importance may be unknown, even to human experts.

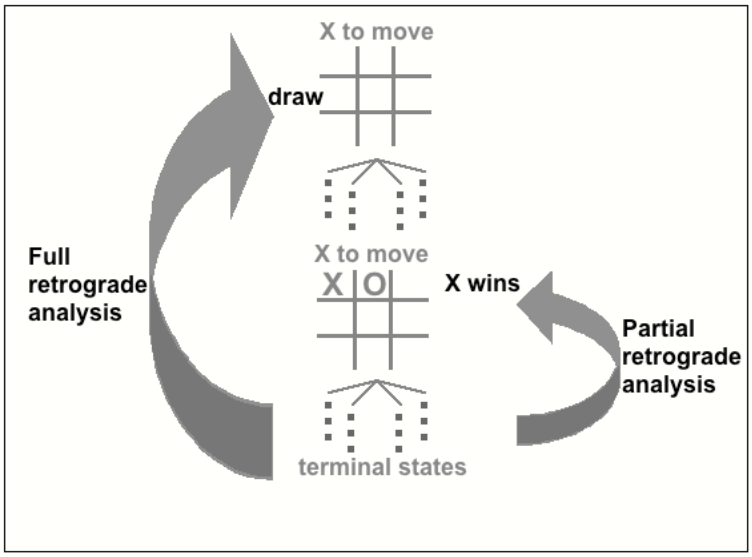


Figure 3: Retrograde analysis backs up rule-determined values.

Search and knowledge

Confronted with a large game tree and without a perfect evaluation function, the typical game-playing program relies instead on heuristic search in the game tree. The program searches several ply down from the current state, labels each game state it reaches with an estimate of its game theoretic value as computed by a heuristic evaluation function, and then backs those values up to select the best move. Most classic game-playing programs devote extensive time and space to such heuristic search. The most successful variations preserve exhaustive search's correctness: a transposition table to save previously evaluated positions, the α-β algorithm to *prune* (not search) irrelevant segments of the game tree, extensions along promising lines of play, and extensions that include forced moves. Other search algorithms take conservative risks; they prune unpromising lines early or seek *quiesence,* a relatively stable heuristic evaluation in a small search tree. Whatever its search mechanisms, however, a powerful game-playing program typically plays only a single game, because it also relies on knowledge.

Knowledge is traditionally incorporated into a game-playing program in three 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, prior to competition, the program calculates the true game-theoretic values of certain nodes with exhaustive search and stores them with their optimal moves (*endgame database*). Because a heuristic evaluation function always returns any available endgame values, the larger that database, the more accurate the evaluation and the better search is likely to perform.

Early attempts at mechanized game playing

Chess has long been the focus of automated game playing. The first known mechanical game player was for a chess endgame (king and rook against king), constructed about 1890 by Torrès y Quevedo. In the 1940's many researchers began to consider how a computer might play chess well, and constructed specialized hardware and algorithms for chess. Work by Shannon, Turing, and de Groot was particularly influential. By 1958 a program capable of playing the entire game was reported, and by the mid-1960's computers had begun to compete against each other in tournaments (Marsland 1990).

At about the same time, Samuel was laying the foundation for today's ambitious game-playing programs, and much of machine learning, with his checkers player (Samuel 1959; Samuel 1967). A game state was summarized by his program in a vector of 38 feature values. The program searched at least 3 ply, with deeper searches for positions associated with piece capture and substantial differences in material. The checker player stored as many evaluated positions as possible, reusing them to make subsequent decisions. Samuel tested a variety of evaluation functions, beginning with a prespecified linear combination of the features. He created a compact representation for game states, as well as a routine to learn weighted combinations of 16 of the features at a time. Samuel's work pioneered rote learning, generalization, and co-evolution. His program employed α-β search, tuned its evaluation function to book games played by checker masters, constructed a library of moves learned by rote, and experimented with non-linear terms through a signature table. After playing 28 contests against itself, the checker program had learned to play tournament-level checkers, but it remained weaker than the best human players.

For many years it was the chess programs that held the spotlight. The first match between two computer programs was played by telegraph in 1967, when a Russian program defeated an American one 3 – 1. Although they initially explored a variety of techniques, the most successful chess programs went on to demonstrate the power of fast, deep game-tree search. These included versions of Kaissa, MacHack 6, Chess 4.6, Belle, Cray Blitz, Bebe, Hitech, and a program named Deep Thought, the precursor of Deep Blue. As computers grew more powerful, so did chess playing programs, moving from Chess 3.0's 1400 rating in 1970 to Deep Blue's championship play in 1997.

Brute force wins the day

Brute force is fast, deep search plus enormous memory directed to the solution of a problem. In checkers and in chess, brute force has triumphed over acknowledged human champions. Both programs had search engines that rapidly explored enormous subtrees, and supported that search with extensive, efficient opening books and endgame databases. Each also had a carefully tuned, human-constructed, heuristic evaluation function, with features whose relative importance were well understood in the human expert community.

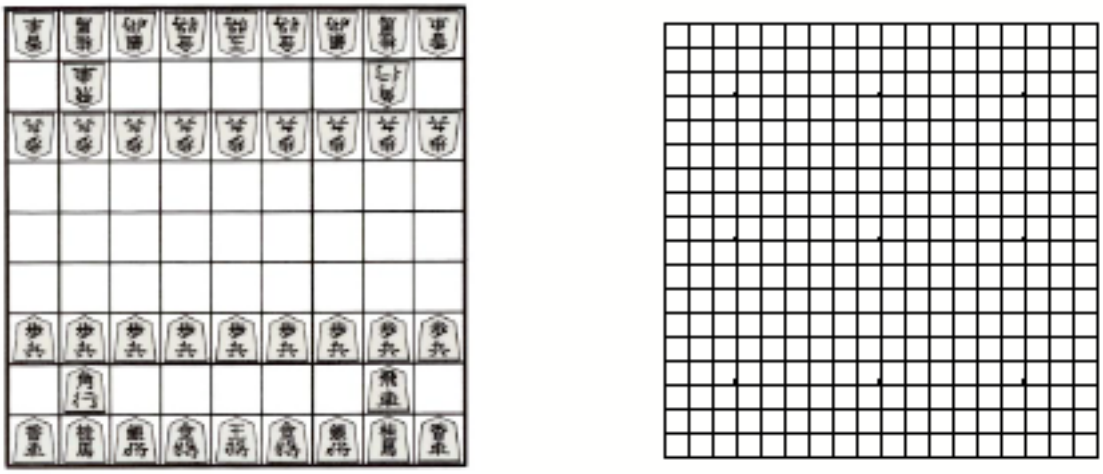
In 1994, Chinook became the world's champion checker player, defeating Marion Tinsley (Schaeffer 1997). Its opening book included 80,000 positions. Its 10-gigabyte

endgame database, constructed by exhaustive forward search, included about 443 billion positions, every position in which no more than 8 pieces (checkers or kings) remain on the board. The frequency with which Chinook's search reached these game-theoretic values was in large measure responsible for the program's success.

In 1997 Deep Blue defeated Garry Kasparov, the human chess champion. Deep Blue's custom chess-searching hardware enabled it to evaluate 200 million moves per second, sometimes to depths over 30 ply. In the year immediately before its victory, the program benefited from a substantial infusion of grandmaster-level knowledge, particularly in its evaluation function and its opening book. Deep Blue's endgame database included all chess positions with five or fewer pieces, but it was rarely reached.

Simulation and machine learning, the alternatives

There are, however, games more difficult than chess, games where programs require more than brute force to win. Consider, for example, shogi and Go, played on the boards in Figures 4(a) and 4(b), respectively. Although the branching factor for chess is 35, for Shogi it is 80 – 150, and for Go it is 250. Such a large branching factor makes deep search intractable. Games with very long contests also reduce the opportunity for search to reach an endgame database, where the evaluation function would be perfect. For example, the typical checkers contest averages about 50 moves, but the typical Go contest averages more than 300. In games that include imperfect information (e.g., a concealed hand of cards) or non-determinism (e.g., dice), the brute force approach represents each possibility as a separate state. Once again, the branching factor makes deep search intractable. In bridge, for example, after bidding the declarer can see 26 cards, but there are more than 20 million ways the other 26 cards may be distributed between the opponents' hands.



(a) (b)

Figure 4: (a) The starting position in Shogi. (b) the Go board.

In a game with a very large branching factor, rather than search all possibilities exhaustively, a program can sample the game tree by simulation. Simulation generates the unknown information (e.g., the opponents' hands) at random and evaluates game

states based on that assumption. Since a single random guess is unlikely to be correct, simulation is repeated, typically thousands of times. The evaluation function is applied to each state resulting from the same move in the simulated trees, and averaged across them to approximate the goodness of a particular move. Simulation can be extended as many ply as desired. Maven, for example, plays Scrabble®, a game in which contestants place one-letter tiles into a crossword format. Scrabble® is non-deterministic because tiles are selected at random, and it involves imperfect information because unplayed tiles are concealed. Nonetheless, Maven is considered the best player of the game, human or machine (Sheppard 1999). Instead of deep search, Maven uses a standard, game-specific move generator (Appel and Jacobson 1988), a probabilistic simulation of tile selection with 3-ply search, and the B* search algorithm in the endgame.

When people lack the requisite expert knowledge, a game-playing program can learn. A program that learns executes code that enables it to process information and reuse it appropriately. Rather than rely upon the programmer's knowledge, such a program instead acquires knowledge that it needs to play expertly, either during competition (*online*) or in advance (*offline*). A variety of learning methods have been directed toward game playing: rote memorization of expert moves, deduction from the rules of a game, and a variety of inductive methods. An approach that succeeds for one game does not necessarily do well on another. Thus a game-learning program must be carefully engineered.

A game-playing program can learn openings, endgame play, or portions of its evaluation function. Openings are relatively accessible from human experts' play. For example, Samuel's checker player acquired a database of common moves online, and Deep Blue learned about grandmasters' openings offline. Endgame database computations are costly but relatively straightforward; Chinook's endgame database, learned offline, was essential to its success. In a game whose large branching factor or lengthy contests preclude deep search, however, an endgame database is rarely reached during lookahead.

Machine learning for game playing often focuses on the evaluation function. TD-gammon is one of the world's strongest backgammon players. The mover in backgammon rolls a pair of dice on every turn; as a result, the branching factor is 400, precluding extensive search. TD-gammon models decision making with a neural network whose weights are acquired with temporal difference learning in millions of contests between two copies of the program. Given a description of the position with human-supplied features the neural net serves as an evaluation function; during competition, TD-gammon uses it to select a move after a 2-to-3-ply search (Tesauro 1995).

Ideally, a program could learn not only weights for its evaluation function, but also the features it references. Logistello plays Othello (Reversi); in 1997 it defeated Takeshi Murakami, the human world champion, winning all 6 contests in the match (Buro 1998). Logistello's heuristic evaluation function is primarily a weighted combination of simple patterns that appear on the board, such as horizontal or diagonal lines. (Which player has the last move and how far a contest has progressed are also included.) To produce this evaluation function, 1.5 million weights for elaborate conjunctions of these features were

calculated with gradient descent during offline training, from analysis of 11 million positions. Although it uses a sophisticated search algorithm and a large opening book, Logistello's evaluation function is the key to its prowess. Its creator supplied the raw material for Logistello's evaluation function, but the program learned features produced from them, and learned weights for those features as well.

Cognition and game-playing programs

Although no person could search as quickly or recall as accurately as a champion program, there are some aspects of these programs that simulate human experts. A good human player remembers previous significant experiences, as if the person had a knowledge base. A good human player expands the same portion of a game tree only once in a contest, as if the person had a transposition table. A good human player has a smaller, but equally significant, opening book and recognizes and employs endgame knowledge.

There are also features of human expertise that programs generally lack. People plan, but planning in game playing has not performed as well as heuristic search. People narrow their choices, but simulation or exhaustive search, at least for the first few ply, have proved more reliable for programs. People construct a model of the opposition and use it to guide decision making, but most programs are oblivious of their opposition. People have a variety of rationales for decisions, and are able to offer explanations for them, but most programs have opaque representations. Skilled people remember *chunks* (unordered static spatial patterns) that could arise during play (Chase and Simon 1973), but, at least for chess programs, heuristic search ultimately proved more powerful. Finally, many people play more than one game very well, but the programs described here can each only play a single game. (One program, Hoyle, learns to play multiple games, but their game trees are substantially smaller than chess'.)

The cognitive differences between people and programs become of interest in the face of games, such as shogi and Go, that programs do not yet play well at all. These games do not yield readily to search. Moreover, the construction of a powerful evaluation function for these games is problematic, since even the appropriate features are unknown. In shogi, unlike chess, there is no human consensus on the relative strength of the individual pieces (Beal and Smith 1998). In Go there are thousands of plausible features (often couched in proverbs) whose interactions are not well understood. Finally, because the endgame is likely to have at least as large a branching factor as earlier positions, the construction of a useful endgame database for either game is intractable. Although both games have attracted many talented researchers and have their own annual computer tournaments, no entry has yet played either game as well as a strong amateur human.

Timed photographs of a chess player's brain demonstrate that that perception is interleaved with cognition (Nichelli, Grafman, Pietrini, Alway, Carton, and Miletich 1994). Although Go masters do not appear to have chunks as originally predicated (Reitman 1976), there is recent evidence that these people do see dynamic patterns and readily annotate them with plans. Moreover, Go players' memories now appear to be cued to sequences of visual perceptions. As a result, despite their inferiority for chess

programs, work in Go continues to focus on patterns and plans. Another promising technique, foreshadowed by the way human experts look at the Go board, is decomposition search, which replaces a single full search with a set of locally restricted ones (Muller 1999).

The challenges presented by popular card games, such as bridge and poker, have also received attention recently. Both involve more than two contestants and include imperfect information. Bridge offers the challenge of pairs of collaborating opponents, while poker permits tacit alliances among the contestants. At least one bridge program has won masters' points in play against people, relying on simulation of the concealed card hands. Poker pits a single contestant simultaneously against many others, each with an individual style of play. Poki plays strong Texas Hold'em poker, although not as well as the best humans. The program bases its bets on probabilities, uses simulation as a search device, and has begun to model its opponents.

Finally, a synergy can develop between game-playing programs and the human experts they simulate. Scrabble® and backgammon both provide examples. Maven has hundreds of human-supplied features in its evaluation function. The program learned weights for those features from several thousand contests played against itself. Since their 1992 announcement, Maven's weights have become the accepted standard for both human and machine players. Meanwhile, TD-gammon's simulations, known as *rollouts*, have become the authority on the appropriateness of certain play. In particular, human professionals have changed their opening style based on data from TD-gammon's rollouts.

Summary

Game-playing programs are powerfully engineered expert systems. They often have special-purpose hardware, and they employ concise representations designed for efficiency. Where the branching factor permits, a game-playing program relies on fast, deep, algorithmic search, guided by heuristics that estimate the value of alternative moves. If that is not possible, simulation is used to determine a decision. Champion programs play a single game, and benefit from vast stores of knowledge, knowledge either provided by people or learned by the programs from their experience. Nonetheless, challenging games remain where humans play best.

References

Appel, A. W. and Jacobson, G. J. 1988. The World's Fastest Scrabble Program. *Communications of the ACM*, 31(5): 572-578.

Beal, D. and Smith, M. 1998. First results from Using Temporal Difference learning in Shogi. In *Proceedings of the First International Conference on Computers and Games*. Tsukuba, Japan.

Billings, D., Pena, L., Schaeffer, J. and Szafron, D. 1999. Using Probabilistic Knowledge and Simulation to Play Poker. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence,* 697-703. .

Buro, M. 1998. From Simple Features to Sophisticated Evaluation Functions. In *Proceedings of the First International Conference on Computers and Games*. Tsukuba.

Chase, W. G. and Simon, H. A. 1973. The Mind's Eye in Chess. In W. G. Chase (Ed.), *Visual Information Processing*, 215-281. New York: Academic Press.

Marsland, T. A. 1990. A Short History of Computer Chess. In T. A. Marsland, & J. Schaeffer (Ed.), *Computers, Chess, and Cognition*, 3-7. New York: Springer-Verlag.

Muller, M. 1999. Decomposition Search: A Combinatorial Games Approach to Game Tree Search, with Applications to Solving Go Endgames. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence,* 578-583. Stockholm: Morgan Kaufman.

Nichelli, P., Grafman, J., Pietrini, P., Alway, D., Carton, J. and Miletich, R. 1994. Brain Activity in Chess Playing. *Nature*, 369: 191.

Reitman, J. S. 1976. Skilled Perception in Go: Deducing Memory Structures from Inter-Response Times. *Cognitive Psychology*, 8: 36-356.

Samuel, A. L. 1959. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3: 210-229.

Samuel, A. L. 1967. Some Studies in Machine Learning Using the Game of Checkers. II - Recent Progress. *IBM Journal of Research and Development*, 11: 601-617.

Schaeffer, J. 1997. *One Jump Ahead: Challenging Human Supremacy in Checkers*. New York: Springer-Verlag.

Sheppard, B. 1999. Mastering Scrabble. *IEEE Intelligent Systems*, 14(6): 15-16.

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon. *CACM*, 38(3): 58-68.

Further Reading

Berlekamp, E. R., Conway, J. H. and Guy, R. K. 1982. *Winning Ways for Your Mathematical Plays*. London: Academic Press.

Conway, J. H. 1976. *On Numbers and Games*. New York: Academic Press.

Holding, D. 1985. *The Psychology of Chess Skill*. Hillsdale, NJ: Lawrence Erlbaum.