A Gamut of Games *

Jonathan Schaeffer
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
jonathan@cs.ualberta.ca

August 26, 2001

Abstract

In 1950 Claude Shannon published his seminal work on how to program a computer to play chess. Since then, developing game-playing programs that can compete with (and even exceed) the human world champions has been a long-sought-after goal of the artificial intelligence research community. In Shannon's time, it would have seemed unlikely that only a scant 50 years would be needed to develop programs that play world-class backgammon, checkers, chess, Othello, and Scrabble. These remarkable achievements are the result of a better understanding of the problems being solved, major algorithmic insights, and tremendous advances in hardware technology. Computer games research is one of the important success stories of artificial intelligence. This article reviews the past successes, current projects, and future research directions for artificial intelligence using computer games as a research test-bed.

1 Introduction

Games are ideal domains for exploring the capabilities of computational intelligence. The rules are fixed, the scope of the problem is constrained, and the interactions of the players are well defined. Contrast this to the real world—the game of life—where the rules often change, the scope of the problem is almost limitless, and the participants interact in an infinite number of ways. Games can be a microcosm of the real world (e.g. the role of game theory in economics, social interaction, and animal behavior), and successfully achieving high computer performance in a nontrivial game can be a stepping stone towards solving more challenging real-world problems.

*Portions of this article have been published in [53] and are reproduced with permission and without further citation.

Historically, games have been a popular choice for demonstrating new research ideas in artificial intelligence (AI). Indeed, one of the early goals of AI was to build a program capable of defeating the human world chess champion in a match. This challenge proved to be more difficult than was anticipated; the AI literature is replete with optimistic predictions. It eventually took almost 50 years to complete the task—a remarkably short time when one considers the software and hardware advances needed to make this amazing feat possible. Often overlooked, however, is that this result was also a testament to human abilities. Considering the formidable computing power that *Deep Blue* used in its 1997 exhibition match against world chess champion Garry Kasparov (machine: 200,000,000 chess positions per second; man: two per second), one can only admire the human champions for withstanding the technological onslaught for so long.

Computer-games research was started by some of the luminaries in computing science history. In 1950, Claude Shannon published his seminal paper that laid out the framework for building high-performance game-playing programs [55]. In 1951, Alan Turing did a hand simulation of his computer chess algorithm (a lack of resources prevented him from actually programming it) [73]; the algorithm lost to a weak human player. Around this time, Arthur Samuel began work on his famous checkers-playing program, the first program to achieve notable success against human opposition [49, 51]. By 1958, Alan Newell and Herb Simon had begun their investigations into chess, which led eventually led to fundamental results for artificial intelligence and cognitive science [43]. An impressive lineup to say the least!

In the half century years since Shannon's paper, enormous progress has been made in constructing high-performance game-playing programs. In Shannon's time, it would have seemed unlikely that within a scant 50 years checkers (8×8 draughts), Othello[1], and Scrabble[2] programs would exist that exceed the abilities of the best human players, while backgammon and chess programs could play at a level comparable to the human world champion. These remarkable accomplishments are the result of a better understanding of the problems being solved, major algorithmic insights, and tremendous advances in hardware technology. The work on computer games has been one of the most successful and visible results of artificial intelligence research. For some games, one could argue that the Turing test has been passed [37].

This article discusses the progress made in developing programs for the classic board and card games. For a number of games, a short history of the progress in building a world-class program for that game is given, along with a brief description of the strongest program. In each case we highlight a single feature of the program that is a major contributor to the program's strength. The histories are necessarily brief. I apologize in advance to the many hard-working researchers and hobbyists whose work is not mentioned here.

Section 2 briefly summarizes some of the major advances in technology that

[1] Othello is a registered trademark of Tsukuda Original, licensed by Anjar Co.

[2] Scrabble is a registered trademark of the Milton Bradley Company, a division of Hasbro, Inc.

facilitated the construction of world-championship-caliber programs. Section 3 reports the past successes where computers have met or exceeded the best human players (backgammon, checkers, chess, Othello, and Scrabble). Section 4 highlights games of current academic interest (bridge, go, and poker). Section 5 discusses some of the future challenges of using games as a research test-bed for artificial intelligence.

Although this article emphasizes the artificial intelligence viewpoint, one should not underestimate the engineering effort required to build these programs. One need only look at the recent success of the *Deep Blue* chess machine to appreciate the effort required. That project spanned eight years (12 if one includes the pre-IBM time), and included several full-time people, extensive computing resources, chip design, and grandmaster consultation. Some of the case studies hint at the amount of work required to construct these systems. In all cases, the successes reported in this article are the result of consistent progress over many years.

2 Enabling Technologies

The biggest advances in computer game-playing have come as a result of work done on the alpha-beta search algorithm. This algorithm received the most attention because of the research community's preoccupation with chess. With the *Deep Blue* victory over world chess champion Garry Kasparov, interest in methods suitable for chess has waned and been replaced by activity in other games. One could argue that the chess victory removed a ball and shackle that was stifling the creativity of researchers who were building high-performance game-playing systems.

The alpha-beta research led to a plethora of search enhancements which significantly improved the efficiency of the search. Some of these enhancements include iterative deepening, caching previously seen sub-tree results (transposition tables), successor reordering, search extensions/reductions, probabilistic cutoffs, and parallel search. The results are truly amazing. Even though there is an exponential difference between the best-case and the worst-case for an alpha-beta search, most high-performance game-playing programs are searching within a small constant of the best case [45].

Sadly, the community of researchers involved in this work have done a relatively poor job of selling the technology, resulting in many of the ideas being reinvented for other domains. For example, many search techniques pioneered with alpha-beta have become standard in other search domains, with few realizing the lineage of the ideas.

At the heart of many game-playing programs is an evaluation function. Early on, game developers quickly encountered the knowledge acquisition bottleneck, and traded-off quality of knowledge for speed of the program. Simple evaluation functions, linear combinations of easily identifiable features, were the mainstay of computer games programs for many decades. Alternative approaches, such as modeling human cognitive processes, turned out to be much harder to do than initially expected, and generally resulted in poor performance. Games programmers quickly learned that a little heuristic knowledge, when combined with deep search, can produce amazing performance results. Indeed, one could argue that the viability of brute-force search, once a term with negative connotations in the AI community, is one of the main research results from games-related research [22].

In the last decade, new techniques have moved to the forefront of games research. Two in particular are mentioned here since they are likely to play a more prominent role in the near future:

1. Monte Carlo simulation has been successfully applied to games with imperfect or non-deterministic information. In these games it is too expensive to search all possible outcomes. Instead only a representative sample is chosen to give a statistical profile of the outcome. This technique has been successful in bridge, poker and Scrabble.

2. Temporal-difference learning is the direct descendent of Samuel's machine learning research [64]. Here a database of games (possibly generated by computer self-play) can be used to bootstrap a program to find a good combination of knowledge features. The algorithm has been successfully applied to backgammon, and has recently shown promise in chess.

The most obvious hardware advance is simply speed. To read about Samuel's checkers-playing program running on a 1963 machine that could execute 15 million additions *per minute* [44] starkly brings home the point that orders of magnitude more computing power makes many things possible. Indeed, considering the paucity of computing power at Samuel's disposal, one can only be filled with admiration at what he achieved.

Computer games research pioneered competitions to assess the quality of the systems being developed. Since 1970, there have been annual computer chess tournaments. There is now an annual Computer Olympiad which brings together many of the top programs and their developers in head-to-head competition (see `www.msoworld.com`). The competitive spirit has spread throughout the AI community; competitions now exist for other applications, including theorem proving, planning and natural language.

3 Success in Computer Games

In a number of games, computers have enjoyed success that puts them on par or better with the best humans in the world. In some sense, this is the past, in that active research to develop high-performance programs for these games is on the wane (or is now non-existent). These include games where computers are better than all humans (checkers, Othello, Scrabble) and those where computers are competitive with the human world champion (backgammon and chess).

3.1 Backgammon

The first concerted effort at building a strong backgammon program was undertaken by Hans Berliner of Carnegie Mellon University. In 1979 his program, *BKG9.8*, played an exhibition match against the the newly-crowned world champion Luigi Villa [6, 7]. The stakes were $5,000, winner take all. The final score was seven points to one in favor of the computer, with *BKG9.8* winning four of the five games played (the rest of the points came from the doubling cube).

Backgammon is a game of both skill and luck. In a short match, the dice can favor one player over another. Berliner writes that "In the short run, small percentage differences favoring one player are not too significant. However, in the long run a few percentage points are highly indicative of significant skill differences" [7]. Thus, assessing the results of a five-game match are difficult. Afterwards Berliner analyzed the program's play and concluded that [6]:

> There is no doubt that *BKG9.8* played well, but down the line Villa played better. He made the technically correct plays almost all the time, whereas the program did not make the best play in eight out of 73 non-forced situations.

BKG9.8 was an important first step, but major work was still needed to bring the level of play up to that of the world's best players.

In the late 1980s, IBM researcher Gerry Tesauro began work on a neural-net-based backgammon program. The net used encoded backgammon knowledge and, training on data sets of games played by expert players, learned the weights to assign to these pieces of knowledge. The program, *Neurogammon*, was good enough to win first place in the 1989 Computer Olympiad [68].

Tesauro's next program, *TD-Gammon* used a neural network that was trained using temporal difference learning. Instead of training the program with data sets of games played by humans, Tesauro was successful in having the program learn using the temporal differences from *self-play* games. The evolution in *TD-Gammon* from version 0.0 to 3.0 saw an increase in the knowledge used, a larger neural net, and the addition of small selective searches. The resulting program is acknowledged to be on par with the best players in the world, and possibly even better.

In 1998, an exhibition match was played between world champion Malcolm Davis and *TD-Gammon* 3.0 (at the AAAI'98 conference). To reduce the luck factor, 100 games were played over three days. The final result was a narrow eight-point win for Davis. Both Davis and Tesauro have done extensive analysis of the games, coming up with similar conclusions [66]:

> While this analysis isn't definitive, it suggests that we may have witnessed a superhuman level of performance by *TD-Gammon*, marred only by one horrible blunder redoubling to 8 in game 16, costing a whopping 0.9 points in equity and probably the match!

A notable feature of *TD-Gammon* is its neural net evaluation function. The net takes as input the current board position and returns as output the score for the position (roughly, the probability of winning) [69]. The net has approximately 300 input values [65, 67]. The latest version, *TD-Gammon* 3.0, contains 160 hidden units. Each unit takes a linear sum of the weighted values of its inputs, and then converts it to a value in the range -3 to 3 (a backgammon is worth three points, a gammon two, and a win, one point). The conversion is done with a sigmoid function, allowing the output to be a nonlinear function of the inputs. The resulting neural net has approximately 50,000 weights that need to be trained.

The weights in the hidden units were trained using temporal difference learning from self-play games. By playing the program against itself, there was an endless supply of data for the program to train itself against. In a given game position, the program uses the neural net to evaluate each of the roughly 20 different ways it can play its dice roll, and then chooses the move leading to the maximum evaluation. Each game is played to completion, and then temporal difference learning is applied to the sequence of moves. Close to 1,500,000 self-play games were used for training *TD-Gammon* 3.0.

Tesauro's success with temporal difference learning in his backgammon program is a major milestone in artificial intelligence research.

3.2 Checkers

Arthur Samuel began thinking about a checkers program in 1948 but did not start coding until a few years later. He was not the first to write a checkers-playing program; Christopher Strachey pre-dated him by a few months [63]. Over the span of three decades, Samuel worked steadily on his program, with performance taking a back seat to his higher goal of creating a program that learned. Samuel's checkers player is best known for its single win against Robert Nealey in a 1963 exhibition match. From this single game, many people erroneously concluded that checkers was a "solved" game.

In the late 1970's, a team of researchers at Duke University built a strong checkers-playing program that defeated Samuel's program in a short match [72]. Early success convinced the authors that their program was possibly one of the 10 best players in the world. World champion Marion Tinsley effectively debunked that, writing that: "The programs may indeed consider a lot of moves and positions, but one thing is certain. They do not see much!" [71]. Efforts to arrange a match between the two went nowhere and the Duke program was quietly retired.

Interest in checkers was rekindled in 1989 with the advent of strong commercial programs and a research effort at the University of Alberta: *Chinook*. *Chinook* was authored principally by Jonathan Schaeffer, Norman Treloar, Robert Lake, Paul Lu, and Martin Bryant. In 1990, the program earned the right to challenge for the human world championship. The checkers federations refused to sanction the match, leading to the creation of a new title: the world man-machine championship. This title was contested for the first time in 1992, with Marion Tinsley defeating *Chinook* in a 40-game match by a score of 4 wins to 2. *Chinook*'s wins were the first against a reigning world champion in a non-exhibition event for any competitive game.

There was a rematch in 1994, but after six games (all draws), Tinsley resigned the match and the title to *Chinook*, citing health concerns. The following week he was diagnosed with cancer, and he died eight months later. *Chinook* has subsequently defended its title twice, and has not lost a game since 1994. The program was retired from human competitions in 1997 [52].

The structure of *Chinook* is similar to that of a typical chess program: search, knowledge, database of opening moves, and endgame databases [52, 54]. *Chinook* uses alpha-beta search with a myriad of enhancements including iterative deepening, transposition table, move ordering, search extensions, and search reductions. *Chinook* was able to average a minimum of 19-ply searches against Tinsley (using 1994 hardware) with search extensions occasionally reaching 45 ply into the tree. The median position evaluated was typically 25-ply deep into the search.

A notable feature in *Chinook* is its use of endgame databases. The databases contain all checkers positions with eight or fewer pieces, 444 billion (4×10^{11}) positions compressed into six gigabytes for real-time decompression. Unlike chess programs which are compute-bound, *Chinook* becomes I/O-bound after a few moves in a game. The deep searches mean that the database is occasionally being hit on the *first move* of a game. The databases introduce accurate values (win/loss/draw) into the search (no error), reducing the program's dependency on its heuristic evaluation function (small error). In many games, the program is able to backup a draw score to the root of a search within 10 moves by each side from the start of a game. This suggests that it may be possible to determine the game-theoretic value of the starting position of the game (one definition of "solving" the game).

Chinook is the first program to win a human world championship for any game. At the time of *Chinook*'s retirement, the gap between the program and the highest-rated human was 200 rating points (using the chess rating scale) [52]. A gap this large means that the program would score 75% of the possible points in a match against the human world champion. Since then, faster processor speeds mean that *Chinook* has become stronger, further widening the gap between man and machine.

3.3 Chess

The progress of computer chess was strongly influenced by an article by Ken Thompson which equated search depth with chess-program performance [70]. Basically, the paper presented a formula for success: build faster chess search engines. The milestones in chess program development become a statement of the state-of-the-art in high-performance computing:

- 1978-1980: The pioneering programs from Northwestern University, most notably Chess 4.6 [60], ran on a top-of-the-line Control Data computer and achieved the first major tournament successes.

- 1980-1982: *Belle*, the first program to earn a U.S. master title, was a machine built to play chess. It consisted of 10 large wire-wrapped boards using LSI chips [15].
- 1983-1984: *Cray Blitz* used a multi-processor Cray supercomputer [30].
- 1985-1986: The *Hitech* chess machine was based on 64 special-purpose VLSI chips (one per board square) [8, 16].
- 1985-1986: *Waycool* used a 256-processor hypercube [17].
- 1987-present: *ChipTest* (and its successors *Deep Thought* and *Deep Blue*) took VLSI technology even further to come up with a chess chip [27, 28, 29].

In 1987, *ChipTest* shocked the chess world by tieing for first place in a strong tournament, finishing ahead of a former world champion and defeating a grandmaster. The unexpected success aroused the interest of world champion Garry Kasparov, who played a two-game exhibition match against the program in 1989. Man easily defeated machine in both games.

The *Deep Blue* team worked for seven years on improving the program, including designing a single-chip chess search engine and making significant strides in the quality of their software. In 1996, the chess machine played a six-game exhibition match against Kasparov. The world champion was stunned by a defeat in the first game, but he recovered to win the match, scoring three wins and two draws to offset the single loss. The following year, another exhibition match was played. *Deep Blue* scored a brilliant win in game two, handing Kasparov a psychological blow from which he never recovered. In the final, decisive game of the match, Kasparov fell into a trap and the game ended quickly. This gave *Deep Blue* an unexpected match victory, scoring two wins, three draws and a loss.

It is important to keep this result in perspective. First, it was an exhibition match; *Deep Blue* did not *earn* the right to play Kasparov.[3] Second, the match was too short to accurately determine the better player; world-championship matches have varied from 16 to 48 games in length. Although it is not clear just how good *Deep Blue* is, there is no doubt that the program is a strong grandmaster.

What does the research community think of the *Deep Blue* result? Many are filled with admiration at this feat of engineering. Some are cautious about the significance. John McCarthy writes that [41]:

> In 1965, the Russian mathematician Alexander Kronrod said, "Chess is the Drosophila[4] of artificial intelligence." However, computer chess has developed much as genetics might have if the geneticists

[3] To be fair, it is unlikely that the international chess federation will ever allow computers to compete for the world championship.

[4] The drosophila is the fruit fly. The analogy is that the fruit fly is to genetics research as games are to artificial intelligence research.

> had concentrated their efforts starting in 1910 on breeding racing Drosophila. We would have some science, but mainly we would have very fast fruit flies.

In retrospect, the chess "problem" turned out to be much harder than was expected by the computing pioneers. The *Deep Blue* result is a tremendous achievement, and a milestone in the history of computing science.

From the scientific point of view, it is to be regretted that *Deep Blue* has been retired, the hardware unused, and the programming team disbanded. The scientific community has a single data point that suggests machine might be better than man at chess. The data is insufficient and the sample size is not statistically significant. Moreover, given the lack of interest in *Deep Blue* from IBM, it is doubtful that this experiment will ever be repeated. Of what value is a single, non-repeatable data point?

Deep Blue and its predecessors represents a decade-long intensive effort by a team of people. The project was funded by IBM, and the principal scientists who developed the program were Feng-hsiung Hsu, Murray Campbell, and Joe Hoane.

The notable technological feature of *Deep Blue* is its amazing speed, the result of building special-purpose chess chips. The chip includes a search engine, a move generator, and an evaluation function [27]. The chip's search algorithm is based on alpha-beta. The evaluation function is implemented as small tables on the chip; the values for these tables can be downloaded to the chip before the search begins. These tables are indexed by board features and the results summed in parallel to provide the positional score.

A single chip is capable of analyzing over two million chess positions per second (using 1997 technology). It is important to note that this speed understates the chip's capabilities. Some operations that are too expensive to implement in software can be done with little or no cost in hardware. For example, one capability of the chip is to selectively generate subsets of legal moves, such as all moves that can put the opponent in check. These increased capabilities give rise to new opportunities for the search algorithm and the evaluation function. Hsu estimates that each chess chip position evaluation roughly equates to 40,000 instructions on a general-purpose computer. If so, then each chip translates to a 100 billion instruction per second chess supercomputer [27].

Access to the chip is controlled by an alpha-beta search algorithm that resides on the host computer (an IBM SP-2). Each of the 32 SP-2 processors could access 16 chips. The reported cumulative performance, 200,000,000 positions analyzed per second, falls short of the peak speed (over one billion positions per second) due to the inherent difficulty of getting good parallel performance out of the alpha-beta algorithm. That massive amount of computing allows the program to search deeper, significantly reducing the probability that it will make an error (as Kasparov found out to his regret).

The artificial intelligence community gave a collective sigh of relief when *Deep Blue* defeated Kasparov. It was time to move on to new challenges in the field.

3.4 Othello

The first major Othello program was Paul Rosenbloom's *Iago* [48], achieving impressive results given its early-1980s hardware. It dominated play against other Othello programs of the time, but played only two games against world-class human players, losing both. The program's ability to predict 59% of the moves played by human experts was extrapolated to conclude that the program's playing strength was of world-championship caliber.

By the end of the decade, *Iago* had been eclipsed. Kai-Fu Lee and Sanjoy Mahajan's program *Bill* represented a major improvement in the quality of computer Othello play [39]. The program combined deep search with extensive knowledge (in the form of precomputed tables) in its evaluation function. Bayesian learning was used to combine the evaluation function features in a weighted quadratic polynomial.

Statistical analysis of the program's play indicated that it was a strong Othello player. *Bill* won a single game against Brian Rose, the highest rated American Othello player at the time. In test games against *Iago*, *Bill* won every game. These results led Lee and Mahajan to conclude that "*Bill* is one of the best, if not the best, Othello player in the world." As usual, there is danger in extrapolating conclusions based on limited evidence.

With the advent of the Internet Othello Server (IOS), computer Othello tournaments became frequent. In the 1990s they were dominated by Michael Buro's *Logistello*. The program participated in 25 tournaments, finished first 18 times, second six times, and fourth once. The program combined deep search with an extensive evaluation function that was automatically tuned. That combined with an extensive database of opening moves and a perfect endgame player are a winning recipe for Othello.

Although it was suspected that by the mid-1990s, computers had surpassed humans in their playing abilities at Othello, this was not properly demonstrated until 1997, when *Logistello* played an exhibition match against world champion Takeshi Murakami. In preparation for the match, Buro writes that [13]:

> *Bill* played a series of games against different versions of *Logistello*. The results showed that *Bill*, when playing 5-minute games running on a PentiumPro/200 PC, is about as strong as a 3-ply *Logistello*, even though *Bill* searches 8 to 9 plies. Obviously, the additional search is compensated for by knowledge. However, the 3-ply *Logistello* can only be called mediocre by today's human standards.
>
> Two explanations for the overestimation of playing strength in the past come to mind: (1) during the last decade human players have improved their playing skills considerably, and (2) the playing strength of the early programs was largely overestimated by using ... non-reliable scientific methods.

Logistello won all six games against Murakami by a total disc count of 264 to 120 [13]. This confirmed what everyone had expected about the relative playing strengths of man and machine. The gap between the best human players and the best computer programs is believed to be large and effectively unsurmountable.

Outwardly, *Logistello* looks like a typical alpha-beta-based searcher. However, the construction of the evaluation function is novel. The program treats the game as having 13 phases: 13–16 discs on the board, 17–20 discs, ..., and 61–64 discs.[5] Each phase has a different set of weights in the evaluation function. The evaluation-function features are patterns of squares comprising combinations of corners, diagonals, and rows. These patterns capture important Othello concepts, such as mobility, stability and parity. *Logistello* has 11 such patterns, which with rotations and reflections yields 46. Some of the patterns include a 3×3 and a 5×2 configuration of stones anchored in a corner, and all diagonals of length greater than 3.

The weights for each entry in each pattern (46) for each phase of the game (11) are determined by linear regression. There are over 1.5 million table entries that need to be determined. The data was trained using 11 million scored positions obtained from self-play games and practice games against another program [12]. The evaluation function is completely table-driven. Given a position, all 46 patterns are matched against the position, with a successful match returning the associated weight. These weights are summed to get the overall evaluation which approximates the final disc differential.

Michael Buro comments on the reasons why *Logistello* easily won the Murakami match[13]:

> When looking at the games of the match the main reasons for the clear outcome are as follows:
>
> 1. Lookahead search is very hard for humans in Othello. The disadvantage becomes very clear in the endgame phase, where the board changes are more substantial than in the opening and middlegame stage. Computers are playing perfectly in the endgame while humans often lose discs.
>
> 2. Due to the automated tuning of the evaluation functions and deep selective searches, the best programs estimate their winning chance in the opening and middlegame phase very accurately. This leaves little room for human innovations in the opening, especially because the best Othello programs are extending their opening books automatically to explore new variations.

3.5 Scrabble

The first documented Scrabble program appears to have been written by Stuart Shapiro and Howard Smith and was published in 1977 [56]. In the 1980s a number of Scrabble programming efforts emerged and by the end of the decade, it was apparent that these programs were strong players. With access to the

[5] Note that there is no need for a phase for less than 13 discs on the board, since the search from the first move easily reaches 13 or more discs.

entire Scrabble dictionary in memory (now over 100,000 words), the programs held an important advantage in any games against humans.

At the first Computer Olympiad in 1989 the Scrabble winner was *Crab* written by Andrew Appel, Guy Jacobson, and Graeme Thomas [40]. Second was *Tyler* written by Alan Frank. Subsequent Olympiads saw the emergence of *TSP* (Jim Homan), which edged out *Tyler* in the second and third Olympiads. All of these programs were very good, and quite possibly strong enough to be a serious test for the best players in the world.

Part of their success was due to the fast, compact Scrabble move generator developed by Appel [4]. Steven Gordon subsequently developed a move generator that was twice as fast, but used five times as much storage [26].

Brian Sheppard began working on a Scrabble program in 1983, and started developing *Maven* in 1986. In a tournament in December 1986, *Maven* scored eight wins and two losses over an elite field, finishing in second place on tie-break. Sheppard describes the games against humans at this tournament [57]:

> *Maven* reels off JOUNCES, JAUNTIER, and OVERTOIL on successive plays, each for exactly 86 points, to come from behind against future national champion Bob Felt. Maven crushed humans repeatedly in offhand games. The human race begins to contemplate the potential of computers.

In the following years, *Maven* continued to demonstrate its dominating play against human opposition. Unfortunately, since it did not compete in the Computer Olympiads, it was difficult to know how strong it was compared to other programs at the time.

In the 1990s, Sheppard developed a pre-endgame analyzer (for when there were a few tiles left in the bag) and improved the program's ability to simulate likely sequences of moves. These represented important advances in the program's ability. It was not until 1997, however, that the opportunity arose to properly assess the program's abilities against world-class players. In 1997, a two-game match between *Maven* and Adam Logan, one of the best players in North America, ended in two wins for the human. Unfortunately, the match was not long enough to get a sense of who was really the best player.

In March 1998, the New York Times sponsored an exhibition match between *Maven* and a team consisting of world champion Joel Sherman and the runner-up Matt Graham. It is not clear whether the collaboration helped or hindered the human side, but the computer won convincingly by a score of six wins to three. The result was not an anomaly. In July of that year, *Maven* played another exhibition match against Adam Logan (at the AAAI'98 conference), scoring nine wins to five.

Shortly after the Logan match, Brian Sheppard wrote:

> The evidence right now is that *Maven* is far stronger than human players. ... I have outright claimed in communication with the cream of humanity that *Maven* should be moved from the "championship caliber" class to the "abandon hope" class, and challenged anyone

> who disagrees with me to come out and play. No takers so far, but maybe one brave human will yet venture forth.

No one has.

Maven divides the game into three phases [59]: early game, pre-endgame, and endgame. The early game starts at move one and continues until there are nine or fewer tiles left in the bag (*i.e.*, with the opponent's seven tiles, this implies that there are 16 or fewer unknown tiles). In the pre-endgame and endgame phases, specialized searches are performed taking advantage of the limited amount of unknown information.

In the early game phase, the program uses simulations to get a statistical analysis of the likely consequences of making a move. Typically, 1,000 three-ply simulations are done when making a move decision. The move leading to the highest average point differential is selected. The issue with the simulations is move generation. On average there are over 700 legal moves per position, and the presence of two blanks in the rack can increase this figure to over 5,000![6] Contrast this, for example, with chess where the average number of moves to consider in a position is roughly 40. Thus, *Maven* needs to pare the list of possible moves down to a small list of likely moves. Omitting an important move from this list will have serious consequences; it will never be played. Consequently, *Maven* employs multiple move generators, each identifying moves that have important features that merit consideration. These move generators are:

- Score and Rack. This generator finds moves that result in a high score and a good rack (tiles remaining in your possession). Strong players evaluate their rack based on the likeliness of the letters being used to aid upcoming words. For example, playing a word that leaves a rack of QXI would be less preferable than leaving QUI; the latter offers more potential for playing the Q effectively.

- Bingo Blocking. Playing all seven letters in a single turn leads to a bonus of 50 points (a *bingo*). This move generator finds moves that reduce the chances of the opponent scoring a bingo on their next turn. Sometimes it is worth sacrificing points to reduce the opponent's chances of scoring big.

- Immediate Scoring. This generates the moves with the maximum number of points (this becomes more important as the end of the game nears).

Each routine provides up to 10 candidate moves. Merging these lists results in typically 20-30 unique candidate moves to consider. In the early part of the game only the Score and Rack generator is used. In the pre-endgame there are four: the three listed above plus a pre-endgame evaluator that "took years to tune to the point where it didn't blunder nearly always" [58]. In the endgame, all possible moves are considered.

The move generation routines are highly effective at filtering the hundreds or thousands of possible moves [58]:

[6] As a frequent Scrabble player, I painfully admit that the number of words that I find are considerably smaller than this!

> It is important to note that simply selecting the one move preferred by the Score and Rack evaluator plays championship caliber Scrabble. My practice of combining 10 moves from multiple generators is evidence of developing paranoia on my part. "Massive overkill" is the centerpiece of *Maven*'s design philosophy.

Obviously, this move filtering works very well, given the level of the program's play. The Scrabble community has extensively analysed *Maven*'s play and found a few minor errors in the program's play. Postmortem analysis of the Logan match showed that *Maven* made mistakes that averaged nine points per game. Logan's average was 40 points per game. *Maven* missed seven fishing moves—opportunities to exchange some tiles (69 points lost), some programming errors (48 points lost), and several smaller mistakes (6 points lost). The programming errors have been corrected. If a future version of *Maven* included fishing, the error rate would drop to less than one point per game. *Maven* would be playing nearly perfect Scrabble.

Of the points lost due to programming errors, Brian Sheppard writes:

> It just drives me crazy that I can think up inventive ways to get computers to act intelligently, but I am not smart enough to implement them correctly.

And that is the soliloquy of every games programmer.

¡¡ INSERT SIDEBAR HERE ¿¿

3.6 Other Games

Superhuman performance has probably been achieved in several lesser-known games. For example, for both the ancient African game of awari (also called mancala) and the recently invented lines of action, there seems little doubt that computers are significantly stronger than all human players [74]. This will not be conclusively demonstrated until the human champions accept the computer challenges for a serious match.

For some games, computers have been able to determine the result of perfect play and a sequence of moves to achieve this result.[7] In these games the computer can play perfectly, in the sense that the program will never make a move that fails to achieve the best-possible result. Solved games include nine men's morris [21], connect-4 [1], qubic [2], go moku [2], and 8×8 domineering [11].

This article has not addressed one-player games (or puzzles). Single-agent search (A*) has been successfully used to optimally solve instances of the 24-puzzle [36] and Rubik's Cube [35].

[7] This is in contrast to the game of Hex where it is easy to prove the game to be a first player win, but computers are not yet able to demonstrate that win.

4 Current Research Efforts

In the past decade, a number of games have become popular research test-beds. These games are resistant to alpha-beta search, either because of the large branching factor in the search tree, or the presence of unknown information. In many respects, the research being done for these games has the potential to be much more widely applicable than the work done on the alpha-beta search-based programs.

4.1 Bridge

Work on computer bridge began in the early 1960s ([5], for example), but it wasn't until the 1980s that major efforts were undertaken. The advent of the personal computer spurred on numerous commercial projects that resulted in programs with relatively poor capabilities. Perennial world champion Bob Hamman once remarked that the commercial programs "would have to improve to be hopeless" [24]. A similar opinion was shared by another frequent world champion, Zia Mahmood. In 1990, he offered a prize of £1,000,000 to the person who developed a program that could defeat him at bridge. At the time, this seemed like a safe bet for the foreseeable future.

In the 1990s, several academic efforts began using bridge for research in artificial intelligence [19, 23, 24, 61, 62]. The commercial *Bridge Baron* program teamed up with Dana Nau and Steve Smith from the University of Maryland. The result was a victory in the 1997 world computer bridge championship. The program used a hierarchical task network for the play of the hand. Rather than building a search tree where each branch was the play of a card, they would define each branch to be a strategy, using human-defined concepts such as finesse and squeeze [61, 62]. The result was an incremental improvement in the program's card play, but it was still far from being world-class caliber.

Beginning in 1998, Matthew Ginsberg's program *GIB* started dominating the computer bridge competition, handily winning the world computer bridge championship. The program started producing strong results in competitions against humans, including an impressive result in an exhibition match against world champions Zia Mahmood and Michael Rosenberg (held at AAAI'98). The match lasted two hours, allowing 14 boards to be played. The result was in doubt until the last hand, before the humans prevailed by 6.31 IMPs (International Match Points). This was the first notable man-machine success for computer bridge-playing programs. Zia Mahmood, impressed by the rapid progress made by *GIB*, withdrew his million pound prize.

GIB was invited to compete in the Par Contest at the 1998 world bridge championships. This tournament tests the contestant's skills at playing out bridge hands. In a select field of 35 of the premier players in the world, the program finished strongly in 12th place. Michael Rosenberg won the event with a score of 16,850 out of 24,000; *GIB* scored 11,210. Of the points lost by *GIB*, 1,000 were due to time (there was a 10 point penalty per minute spent thinking), 6,000 were due to *GIB* not understanding the bidding, and 6,000 were due to *GIB*'s inability to handle some hands where the correct strategy involves combining different possibilities [24].

The name *GIB* originally stood for "Goren In a Box", a tribute to one of the pioneers of bridge. Another interpretation is "Ginsberg's Intelligent Bridge." The current version of *GIB* uses a fast search to play out a hand. It simulates roughly 50 different scenarios for the placement of the opponent's cards, and chooses the play that maximizes the expected score [24]. Ginsberg has developed a new version of the algorithm that will eliminate the simulations and replace it with perfect information [25] [8]. Regardless, *GIB* is very strong at the play of the hand.

A challenging component of the game is the bidding. Most previous attempts at bridge bidding have been based on an expert-defined set of rules. This is largely unavoidable, since bidding is an agreed-upon convention for communicating card information. *GIB* takes this one step further, building on the ability to quickly simulate a hand [24]. The program has access to a large database of bidding rules (7,400 rules from the commercial program *Meadowlark Bridge*). At each point in the bidding, *GIB* queries the database to find the set of plausible bids. For each bid, the rest of the auction is projected using the database, and then the play of the resulting contract is simulated. *GIB* chooses the bid that leads to the average best result for the program.

Although intuitively appealing, this approach does have some problems. Notably the database of rules may have gaps and errors in it. Consider a rule where the response to the bid 4♠ is incorrect in the database. *GIB* will direct its play towards this bid because it assumes the opponents will make the (likely bad) database response. As Ginsberg writes, "it is difficult to distinguish a good choice that is successful because the opponent has no winning options from a bad choice that *appears* successful because the heuristic fails to identify such options" [24].

GIB uses three partial solutions to the problem of an erroneous or incomplete bidding system. First, the bidding database can be examined by doing extensive off-line computations to identify erroneous or missing bid information. This is effective, but can take a long time to complete. Second, during a game, simulation results can be used to identify when a database response to a bid leads to a poor result. This may be evidence of a database problem, but it could also be the result of effective disruptive bidding by *GIB*. Finally, *GIB* can be biased to make bids that are "close" to the suggested database bids, allowing the program the flexibility to deviate from the database.

To summarize, *GIB* is well on the way to becoming a world-class bridge player. The program's card play is already at a world-class level (as evidenced by the Par Contest result), and current efforts will only enhance this. The bidding needs improvement, and this is currently being addressed. Had Zia Mahmood not withdrawn his offer, he might have lost his money within a couple of years from now.

[8] At the time of this writing, these results have not yet been published.

4.2 Go

The history of computer go has not been dominated by hardware advances, as seen in computer chess. Computer go tournaments proliferated in the 1990s, and the organizers had the benefit of the chess experience. Two tournament rules were instituted that had a significant impact on how program development would occur. The first required all competitors to run on a commercially available single-processor machine. This had the advantage of putting all the programs on a level playing field by factoring out most hardware differences. The second rule required that an entire game had to completed in 30 minutes per player. Since games could be as long as 180 moves a side, programmers were faced with critical cost-benefit decisions in their implementations. The rules had the advantages of making tournaments easy to organize (no expensive hardware setup or modem connections needed) and ensured that competitions could be completed quickly with lots of games being played.

The first go program was written by Al Zobrist in 1970 [76]. Walter Reitman and Bruce Wilcox began researching go programs in 1972 [47], an effort that has continued for Wilcox to the current day. These early efforts produced weak programs; there was no obvious single algorithm to build a program around, as alpha-beta had done for chess. The difficulty of writing a go program became evident; a strong program would need lots of patterns and knowledge, with only a limited dependence on search.

Computer go tournaments began in 1984 with a short-lived series of annual tournaments at the USENIX conference. In 1987, the First International Go Congress was held, and there have been annual events ever since. The mid-1990s were dominated by the program *HandTalk*, written by Zhixing Chen. *HandTalk* remained stagnant for a few years while it was being rewritten. During that period, Michael Reiss' Go4++ assumed front-runner status. Although the top programs claim a performance level of up to 3 kyu on the go rating scale (a middle amateur level), most experts believe that the programs are much weaker than that (around 8 kyu).

The Ing Prize has been set up as an incentive to build strong go programs. The grand prize of roughly $1.5 million will be won by the developers of the first program to beat a strong human player on a 19×19 board. To qualify to play for the grand prize, a program must win a number of matches of increasing difficulty. Currently, the programs have to defeat three junior players (ages 11 to 13). Don't let their age fool you; they are very strong players! The winner of the annual International Go Congress gets the chance to play. To qualify for this event, a program must finish in the top three in one of the North American, European, or Asian championships.

Go has been resistant to the techniques that have been successfully applied to the games discussed in this article. For example, because of the 19×19 board and the resulting large branching factor, alpha-beta search alone has no hope of producing strong play. Instead, the programs perform small, local searches that use extensive application-dependent knowledge. David Fotland, the author of the *Many Faces of Go* program, identifies over 50 major components needed by a strong go-playing program. The components are substantially different from each other, few are easy to implement, and all are critical to achieving strong play. In effect, you have a linked chain, where the weakest link determines the overall strength.

Martin Müller (author of *Explorer*) gives a stark assessment of the reality of the current situation in developing go programs [42]:

> Given the complexity of the task, the supporting infrastructure for writing go programs should offer more than is offered for other games such as chess. However, the available material (publications and source code) is far inferior. The playing level of publicly available source code ..., though improved recently, lags behind that of the state-of-the-art programs. Quality publications are scarce and hard to track down. Few of the top programmers have an interest in publishing their methods. Whereas articles on computer chess or general game-tree search methods regularly appear in mainstream AI journals, technical publications on computer go remain confined to hard to find proceedings of specialized conferences. The most interesting developments can be learned only by direct communication with the programmers and never get published.

Although progress has been steady, it will take many decades of research and development before world-championship-caliber go programs exist.

4.3 Poker

There are many popular poker variants. Texas Hold'em is generally acknowledged to be the most strategically complex variant of poker that is widely played. It is the premier event at the annual World Series of Poker.[9] Until recently, poker has been largely ignored by the computing academic community. There are two main approaches to poker research [9]. One approach is to use simplified variants that are easier to analyze. However, one must be careful that the simplification does not remove challenging components of the problem. For example, Findler worked on and off for 20 years on a poker-playing program for 5-card draw poker [18]. His approach was to model human cognitive processes and build a program that could learn, ignoring many of the interesting complexities of the game.

The other approach is to pick a real variant, and investigate it using mathematical analysis, simulation, and/or ad-hoc expert experience. Expert players with a penchant for mathematics are usually involved in this approach. None of this work has led to the development of strong poker-playing programs.

There is one event in the meager history of computer poker that stands out. In 1984 Mike Caro, a professional poker player, wrote a program that he called *Orac* (Caro spelled backwards). It played one-on-one, no-limit Texas Hold'em.

[9] The 2000 winner of this event was Chris Ferguson, whose research career began in artificial intelligence (he has published with Richard Korf [46]).

Few technical details are known about *Orac* other than it was programmed on an Apple II computer in Pascal. However, Caro arranged a few exhibitions of the program against strong players [14]:

> It lost the TV match to casino owner Bob Stupak, but arguably played the superior game. The machine froze on one game of the two-out-of-three set when it had moved all-in and been called with its three of a kind against Stupak's top two pair. Under the rules, the hand had to be replayed. In the [world series of poker] matches, it won one (from twice world champion Doyle Brunson — or at least it had a two-to-one chip lead after an hour and a quarter when the match was cancelled for a press conference) and lost two (one each to Brunson and then-reigning world champion Tom McEvoy), but — again — was fairly unlucky. In private, preparatory exhibition matches against top players, it won many more times than it lost. It had even beaten me most of the time.

Unfortunately, *Orac* was never properly documented and the results never reproduced. It is highly unlikely that *Orac* was as good as this small sample suggests. No scientific analysis was done to see whether the results were due to skill or luck. As further evidence, none of the present day commercial efforts can claim to be anything but intermediate-level players.

In the 1990s, the creation of an Internet Relay Chat poker server gave the opportunity for humans (and computers) to play interactive games over the Internet. A number of hobbyists developed programs to play on IRC. Foremost among them is *R00lbot*, developed by Greg Wohletz. The program's strength comes from using expert knowledge at the beginning of the game, and doing simulations for subsequent betting decisions.

The University of Alberta program *Loki*, authored by Darse Billings, Aaron Davidson, Jonathan Schaeffer and Duane Szafron, is the first serious academic effort to build a strong poker-playing program. *Loki* plays on the IRC poker server and, like *R00lbot*, is a consistent big winner. Unfortunately, since these games are played with fictitious money, it is hard to extrapolate these results to casino poker.

To play poker well, a program needs to be able to assess hand strength (chances that you have the current best hand), assess hand potential (chances that additional cards will improve your hand), model the opponents (exploiting tendancies in their play), handle deception (misleading informaiton given by the opponents), and bluff (deceive the opponents). In strategic games like chess, the performance loss by ignoring opponent modeling is small, and hence it is usually ignored. In contrast, not only does opponent modeling have tremendous value in poker, it can be the distinguishing feature between players at different skill levels. If a set of players all have a comparable knowledge of poker fundamentals, the ability to alter decisions based on an accurate model of the opponent may have a greater impact on success than any other strategic principle.[10]

[10]The importance of opponent modelling can be seen in the First and Second International

To assess a hand, *Loki* compares its cards against all possible opponent holdings. Naively, one could treat all opponent hands as equally likely, however this skews the hand evaluations compared to more realistic assumptions. Many weak hands are likely to have been folded early on in the game. Therefore, for each possible opponent hand, a probability (or weight) is computed that indicates the likelihood that the opponent would have played that hand in the observed manner.

The simplest approach to determining these weights is to treat all opponents the same, calculating a single set of weights to reflect reasonable behavior, and use them for all opponents. An off-line simulation was used to compute the expected value for each possible hand; these results closely approximate the ranking of hands by strong players. This is called Generic Opponent Modeling (GOM) [10]. Although rather simplistic, this model is quite powerful in that it does a good job of skewing the hand evaluations to take into account the most likely opponent holdings.

Obviously, treating all opponents the same is clearly wrong; each player has a different style. Specific Opponent Modeling (SOM) customizes the calculations to include opponent-specific information. The probability of an opponent holding a particular hand is adjusted by feeding into a neural net the betting frequency statistics gathered on that opponent from previous hands. These statistics usually provide enough information to differentiate, for example, aggressive playing styles from conservative ones.

In competitive poker, opponent modeling is much more complex than portrayed here. For example, players can act to mislead their opponents into constructing an erroneous model. Early in a session a strong poker player may try to create the impression of being very conservative, only to exploit that image later in that session when the opponents are using an incorrect opponent model. A strong player has to have a model of each opponent that can quickly adapt to changing playing styles.

At best, *Loki* plays at the strong intermediate level. A considerable gap remains to be overcome before computers will be as good as the best human players. Recent research has focussed on trying to build "optimal" playing strategies [34].

4.4 Other Games

Several less well-known games are providing interesting challenges. The following three examples all have one property in common: a large branching factor.

Shogi, often referred to as Japanese chess, is very popular in Japan, with major tournaments each year culminating in world championsip match. From the search point of view, Shogi is more challenging than chess: 9×9 board (versus 8×8 for chess), 40 pieces (32 for chess), 8 piece types (6), 80-120 average branching factor (40), and captured pieces can return to the board (removed from the board). Checkmating attacks are critical in Shogi; the programs need

RoShamBo (rock, paper scissors) competitions (`www.cs.ualberta.ca/~games`).

specialized checkmate solvers. These solvers have had some spectacular successes. For example, programs are now capable of solving composed problems with a solution length of over 1,500 ply! Nevertheless, the best programs play at the master's level, while world-championship-level play is still a few decades away [31].

Hex is an elegant game with a simple rule set: alternate placing a stone of your colour on an empty square. One player tries to create a chain of stones connecting the top to the bottom of the board. The other player tries to connect the left side to the right side. It can be mathematically shown that the game is a first player win, and that draws are not possible. *Queenbee* was the first program to achieve success against strong programs [75]. The program uses alpha-beta search with a novel evaluation function. *Hexy* is currently the strongest program in the world and is competitive with strong human players for smaller board sizes. The program uses a specialized search for *virtual connections*, using a theorem-prover-like technique for proving that two points not connected can be connected by a series of moves [3].

A recently invented game that has become popular for games researchers is Amazons. It is played on a 10×10 board, with each player having four queens. Pieces move like a queen in chess, but after moving they shoot an arrow in any direction. The square on which the arrow lands now becomes a wall and cannot be occupied by a queen. In effect, each move reduces the playing area available. If you run out of moves, you lose. In the opening phase of the game, there can be several thousand moves to choose from. The best programs typically search five ply ahead (deeper in the endgame). Because of the territorial nature of the game, Amazons is often touted as a research stepping stone between the search-intensive approaches used in chess and the knowledge-intensive approaches used in go. AI research into this game is only three years old. The best programs play reasonably well, but are not yet competitive with strong human players [74].

Interesting research is also being done on puzzles. Recently, major advances have occurred in building programs that can solve crossword puzzles. *Proverb* (Michael Littman, Greg Keim, *et al.*) scores remarkably well (over 95% of the words correct) on the New York Times crossword puzzles *without understanding the clues* [33]!

Another challenging puzzle is Sokoban. Here the large branching factor (could be over 100) and deep solution lengths (some optimal solutions are over 700 moves) make for a daunting search. On a standard test set, the program *Rolling Stone* can only solve 57 of 90 problems [32].

5 The Future of Computer Games

In the realm of board and card games, go will continue to taunt AI researchers for many decades to come. As well, new games will come along to provide interesting challenges. For example, the game of Octi was invented to be resistant to computer algorithms (`www.octi.net`). It is characterized by having a large branching factor, making deep search impractical. However Octi has the additional dimension that a move can change the capabilities of a piece, making it challenging to design an evaluation function.

The research into board and card games is, in some sense, historically motivated because these were interesting challenges at the dawn of the computing age. However, with the advent of home computers, new forms of computer games and a $20 billion (and growing) industry has emerged: interactive computer games. There are numerous products on the market covering the gamut of action games (e.g. shoot'em-up games like Quake), role-playing games (e.g. player goes on a quest, as in Baldur's Gate), adventure games (e.g. navigating through a scripted story, as in King's Quest), strategy games (e.g. controlling armies in a war, such as Command and Conquer), "God" games (e.g. evolving a simulated population, as in SimCity), and sports (e.g. controlling a player or coaching a team, such as FIFA'01) [38]. Historically, these games have been long on graphics, and short on artificial intelligence.[11]

John Laird has promoted interactive computer games as an opportunity for the AI research community [38]. Many interactive computer games require computer characters that need to interact with the user in a realistic, believable manner. Computer games are the ideal application for developing human-level AI. There is already a need for it, since human game players are generally dissatisfied with computer characters. The characters are shallow, too easy to predict, and, all too often, exhibit artificial stupidity rather than artificial intelligence. This has led to the success of on-line games (such as *Ultima Online*), where players compete against other humans. The current state of the art in developing realistic characters can be described as being primitive, with simple rule-based systems and finite-state machines being the norm. The lack of sophistication is due to the lack of research effort (and, cause and effect, research dollars). This is changing, as more games companies and researchers recognize that AI will play an increasingly important role in game design and development. The quality of the computer graphics may draw you to a product, but the play of the game will keep you using the product (and buying the sequel). Artificial intelligence is critical to creating a satisfying gaming experience.

Finally, the last few years have seen research on team games become popular. The annual RoboCup competition encourages hardware builders and software designers to test their skills on the soccer field (`www.robocup.com`).

Although this article has emphasized building games programs that can compete with humans, there are many other AI challenges that can use games as an interesting experimental test bed. Some sample projects include:

1. Data mining: There are large databases of endgame positions for chess, checkers and awari. It is dissatisfying that all a program can do is look up a specific position in the database. If the *exact* positon is in the database, you get useful information, else nothing. Surely there must be some way of mining this data to learn the principals of strong endgame play. As well,

[11] For example, path finding is a critical component of many games, yet it took until 1996 for the industry to "discover" A*.

there are large databases of chess opening moves. Can this be analyzed to discover new opening ideas? Can one characterize opponent's strengths and weaknesses? Can the data be extrapolated to similar positions?

2. Learning: Using temporal-difference learning to tune an evaluation function is just the precursor to other exciting applications of learning technology to games. For example, research in applying learning algorithms can result in more focussed and informed game-tree searches, better opponent modelling in poker, and adaptive characters in commercial games.

3. Annotating games: Developing annotators that can provide an interesting and informative analysis of a game is a challenging problem. There have been some attempts at automating the commentary for chess games (the International Computer Chess Association has an annual competition), but the results are mediocre. It is hard to differentiate between the trivial and the interesting, the verbose and the informative, all the while anticipating the questions humans would like answered in the commentary. An interesting example is the work done on providing computer commentary to RoboCup games [20].

Games will continue to be an interesting domain for exploring new ideas in artificial intelligence.

6 Conclusions

Shannon, Turing, Samuel, Newell and Simon's early writings were pioneering, realizing that computer games could be a rich domain for exploring the boundaries of computer science and artificial intelligence. Software and hardware advances have led to significant success in building high-performance game-playing programs, resulting in milestones in the history of computing. With it has come a change in people's attitudes. Whereas in the 1950s and 1960s, understanding how to build strong game-playing program was at the forefront of artificial-intelligence research, today it has been demoted to lesser status. In part this is an acknowledgment of the success achieved in this field — no other area of artificial intelligence research can claim such an impressive track record of producing high-quality working systems. But it is also a reflection on the nature of artificial intelligence itself. It seems that as the solution to problems become understood, the techniques become less "AIish".

The work on computer games has resulted in advances in numerous areas of computing. One could argue that the series of computer-chess tournaments that began in 1970 and continue to this day represents the longest running experiment in computing science history. Research using games has demonstrated the benefits of brute-force search, something that has become a widely accepted tool for a number of search-based applications. Many of the ideas that saw the light of day in game-tree search have been applied to other algorithms. Building world-championship-caliber games programs has demonstrated the cost of constructing high-performance artificial-intelligence systems. Games have been used as experimental test beds for many areas of artificial intelligence. And so on.

Arthur Samuel's concluding remarks from his 1960 paper are as relevant today as they were when he wrote the paper [50]:

> Just as it was impossible to begin the discussion of game-playing machines without referring to the hoaxes of the past, it is equally unthinkable to close the discussion without a prognosis. Programming computers to play games is but one stage in the development of an understanding of the methods which must be employed for the machine simulation of intellectual behavior. As we progress in this understanding it seems reasonable to assume that these newer techniques will be applied to real-life situations with increasing frequency, and the effort devoted to games ... will decrease. Perhaps we have not yet reached this turning point, and we may still have much to learn from the study of games.

7 Acknowledgments

I would like to extend my deepest admiration to the brave human champions who accepted the challenge of a computer opponent. In most cases, the champion had little to gain, but everything to lose. Malcolm Davis, Garry Kasparov, Adam Logan, Zia Mahmood, Marion Tinsley, Michael Rosenberg, and Takeshi Murakami made it possible to scientifically measure the progress of game-playing programs.

The initial impetus for this article came almost two years ago when Marvin Zelkowitz suggested I write an article for *Advances in Computers 50* reflecting back on the the 40 years since Arthur Samuel wrote an article on computer games in volume 1 of that series. This was eventually worked into a talk that was presented at AAAI'00. I want to thank David Leake for encouraging me to write this article.

Financial support was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] V. Allis. *A Knowledge-Based Approach to Connect-Four. The Game is Solved: White Wins.* M.Sc. thesis, Vrije Universiteit, The Netherlands, 1988.

[2] V. Allis. *Searching for Solutions in Games and Artificial Intelligence.* PhD thesis, University of Limburg, The Netherlands, 1994.

[3] V. Anshelevich. The game of hex: An automatic theorem proving approach to game programming. In *AAAI National Conference*, pages 189–194, 2000.

[4] A. Appel and G. Jacobson. The world's fastest Scrabble program. *Communications of the ACM*, 31(5):572–578, 585, 1988.

[5] E. Berlekamp. A program for playing double-dummy bridge problems. *Journal of the ACM*, 10(4):357–364, 1963.

[6] H. Berliner. Backgammon computer program beats world champion. *Artificial Intelligence*, 14:205–220, 1980.

[7] H. Berliner. Computer backgammon. *Scientific American*, 242(6):64–72, 1980.

[8] H. Berliner and C. Ebeling. Pattern knowledge and search: The SUPREME architecture. *Artificial Intelligence*, 38(2):161–198, 1989.

[9] D. Billings. *Computer Poker.* M.Sc. thesis, University of Alberta, Canada, 1995.

[10] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *AAAI National Conference*, pages 493–499, 1998.

[11] D. Breuker, J. Uiterwijk, and J. van den Herik. Solving 8x8 domineering. *Theoretical Computer Science*, 20:195–206, 2000.

[12] M. Buro. Statistical feature combination for the evaluation of game positions. *Journal of Artificial Intelligence Research*, 3:373–382, 1995.

[13] M. Buro. The Othello match of the year: Takeshi Murakami vs. Logistello. *Journal of the International Computer Chess Association*, 20(3):189–193, 1997.

[14] M. Caro. Email message, March 13, 1999.

[15] J. Condon and K. Thompson. Belle chess hardware. In M. Clarke, editor, *Advances in Computer Chess 3*, pages 45–54. Pergamon Press, 1982.

[16] C. Ebeling. *All the Right Moves.* MIT Press, 1987.

[17] E. Felten and S. Otto. A highly parallel chess program. In *Conference on Fifth Generation Computer Systems*, 1988.

[18] N. Findler. Studies in machine cognition using the game of poker. *Communications of the ACM*, 20(4):230–245, 1977.

[19] I. Frank. *Search and Planning under Incomplete Information: A Study Using Bridge Card Play.* Springer Verlag, 1998.

[20] I. Frank, K. Tanaka-Ishii, H. Okuno, Ỹ. Nakagawa, K. Maeda, K. Nakadai, and H. Kitano. And the fans are going wild! SIG plus MIKE. In *Fourth International Workshop on RoboCup*. Springer-Verlag, 2000. To appear.

[21] R. Gasser. *Efficiently Harnessing Computational Resources for Exhaustive Search*. PhD thesis, ETH Zürich, Switzerland, 1995.

[22] M. Ginsberg. Do computers need common sense? In *Knowledge Representation*, pages 620–626, 1996.

[23] M. Ginsberg. Partition search. In *AAAI National Conference*, pages 228–233, 1996.

[24] M. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *International Joint Conference on Artificial Intelligence*, pages 584–589, 1999.

[25] M. Ginsberg, 2000. Personal communication.

[26] S. Gordon. A faster Scrabble move generation algorithm. *Software Practice and Experience*, 24(2):219–232, 1994.

[27] F. Hsu. IBM's Deep Blue chess grandmaster chips. *IEEE Micro*, (March-April):70–81, 1999.

[28] F. Hsu, T. Anantharaman, M. Campbell, and A. Nowatzyk. Deep Thought. In T. Marsland and J. Schaeffer, editors, *Computers, Chess, and Cognition*, pages 55–78. Springer Verlag, 1990.

[29] F. Hsu, T. Anantharaman, M. Campbell, and A. Nowatzyk. A grandmaster chess machine. *Scientific American*, 263(4):44–50, 1990.

[30] R. Hyatt, A. Gower, and H. Nelson. Cray Blitz. In T. Marsland and J. Schaeffer, editors, *Computers, Chess and Cognition*, pages 111–130. Springer Verlag, 1990.

[31] H. Iida, M. Sakuta, and J. Rollason. The state of the art in computer shogi. *Artificial Intelligence*, 2001. To appear.

[32] A. Junghanns and J. Schaeffer. Domain-dependent single-agent search enhancements. In *International Joint Conference on Artificial Intelligence*, pages 570–575, 1999.

[33] G. Keim, N. Shazeer, M. Littman, S. Agarwal, C. Cheves, J. Fitzgerald, J. Grosland, F. Jiang, S. Pollard, and K. Weinmeister. Proverb: The probabilistic cruciverbalist. In *AAAI National Conference*, pages 710–717, 1999.

[34] D. Koller and A. Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1):167–215, 1997.

[35] R. Korf. Finding optimal solutions to Rubik's Cube using pattern databases. In *AAAI National Conference*, pages 700–705, 1997.

[36] R. Korf. Recent progress in the design and analysis of admissible heuristic functions. In *AAAI National Conference*, pages 1165–1170, 2000.

[37] M. Krol. Have we witnessed a real-life Turing test. *Computer*, 32(3):27–30, 1999.

[38] J. Laird and M. van Lent. Human-level AI's killer application: Interactive computer games. In *AAAI National Conference*, pages 1171–1178, 2000.

[39] K-F. Lee and S. Mahajan. The development of a world class Othello program. *Artificial Intelligence*, 43(1):21–36, 1990.

[40] D. Levy and D. Beal, editors. *Heuristic Programming in Artificial Intelligence*. Ellis Horwood, 1987.

[41] J. McCarthy. AI as sport. *Science*, 276(June 6):1518–1519, 1997.

[42] M. Müller. Computer go: A research agenda. *Journal of the International Computer Chess Association*, 22(2):104–112, 1999.

[43] A. Newell, J. Shaw, and H. Simon. Chess-playing programs and the problem of complexity. *IBM Journal of Research and Development*, 2:320—335, 1958.

[44] J. Pfeiffer. Man vs machine in the mechanical age. *Popular Mechanics*, August:52–57,172–173, 1964.

[45] A. Plaat, J. Schaeffer, W. Pijls, and A. de Bruin. Exploiting graph properties of game trees. In *AAAI National Conference*, pages 234–239, 1996.

[46] C. Powley, C. Ferguson, and R. E. Korf. Depth-first heuristic search on a SIMD machine. *Artificial Intelligence*, 60(2):199–242, 1993.

[47] W. Reitman, J. Kerwin, R. Nado, J. Reitman, and B̃. Wilcox. Goals and plans in a program for playing go. In *ACM National Conference*, pages 123–127, 1974.

[48] P. Rosenbloom. A world-championship-level Othello program. *Artificial Intelligence*, 19(3):279–320, 1982.

[49] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.

[50] A. Samuel. Programming computers to play games. In F. Alt, editor, *Advances in Computers*, volume 1, pages 165–192, 1960.

[51] A. Samuel. Some studies in machine learning using the game of checkers: Recent progress. *IBM Journal of Research and Development*, 11:601–617, 1967.

[52] J. Schaeffer. *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer Verlag, 1997.

[53] J. Schaeffer. The games computers (and people) play. In M. Zelkowitz, editor, *Advances in Computers 50*, pages 189–266. Academic Press, 2000.

[54] J. Schaeffer, J. Culberson, N. Treloar, B. Knight, P. Lu, and D. Szafron. A world championship caliber checkers program. *Artificial Intelligence*, 53(2–3):273–290, 1992.

[55] C. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[56] S. Shapiro and H. Smith. A Scrabble crossword game-playing program. Technical Report 119, State University of New York at Buffalo, Department of Computer Science, 1977.

[57] B. Sheppard. Email message, March 9, 1999.

[58] B. Sheppard. Email message, June 1, 1999.

[59] B. Sheppard. Computer Scrabble. *Artificial Intelligence*, 2001. To appear.

[60] D. Slate and L. Atkin. *Chess Skill in Man and Machine*, chapter Chess 4.5 - The Nortwestern University Chess Program, pages 82–118. Springer-Verlag, 1977.

[61] S. Smith, D. Nau, and T. Throop. Computer bridge: A big win for AI planning. *AI Magazine*, 19(2):93–105, 1998.

[62] S. Smith, D. Nau, and T. Throop. Success in spades: Using AI planning techniques to win the world championship of computer bridge. *AAAI National Conference*, pages 1079–1086, 1998.

[63] C. Strachey. Logical or non-mathematical programmes. *Proceedings of the Association for Computing Machinery Meeting*, pages 46–49, 1952.

[64] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[65] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[66] G. Tesauro. Email message, August 14, 1998.

[67] G. Tesauro. Email message, May 28, 1999.

[68] G. Tesauro. Neurogammon wins computer olympiad. *Neural Computation*, 1:321–323, 1989.

[69] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[70] K. Thompson. Computer chess strength. In M. Clarke, editor, *Advances in Computer Chess 3*, pages 55–56. Pergamon Press, 1982.

[71] M. Tinsley. Letter to the editor. *Scientific American*, (August), 1980.

[72] T. Truscott. The Duke checkers program. *Journal of Recreational Mathematics*, 12(4):241–247, 1979-1980.

[73] A. Turing. Digital computers applied to games. In B. Bowden, editor, *Faster than Thought*, pages 286–295. Pitman, 1953.

[74] J. van den Herik. The Fifth Computer Olympiad. *International Computer Games Association Journal*, 23(3):164–187, 2000.

[75] Jack van Rijswijk. *Computer Hex: Are Bees Better Than Fruit Flies?* M.Sc. thesis, University of Alberta, Canada, 2000.

[76] A. Zobrist. *Feature Extractions and Representation for Pattern Recognition and the Game of Go*. PhD thesis, University of Wisconsin, 1970.

Sidebar: So, You Think You Are Good at Scrabble?

In AAAI'98, *Maven* played an exhibition match against Adam Logan, one of the top Scrabble players in North America. Logan won three of the first four games of the match, but *Maven* won six of the next seven. Going into the critical 12th game, *Maven* led by a score of seven wins to four. The following annotations are based on comments from Brian Sheppard. The columns of a Scrabble board are specified from left-to-right by the letters *a* to *o*. Rows are specified from top-to-bottom using the numbers 1 to 15. Moves are specified by giving the square of the first letter of the word. If the coordinate begins with a number, then the word is placed horizontally. If the coordinate begins with a letter, then the word is placed vertically. The blank is referred to by "?".

Follow along yourself. How good are the moves that you find?

Maven versus Adam Logan

1. **Maven**(ACNTVYZ) plays CAVY at 8f, 24 pts, *Maven*=24 Logan=0. The alternative is ZANY, scoring 32 points, but leaving a poor selection of letters in the rack.

2. **Logan**(EGLNORY) plays YEARLONG at g6, 66 pts, *Maven*=24 Logan=66. The only bingo! A 50 point bonus.

3. **Maven**(ADNNOTZ) plays DOZY at 6d, 37 pts, *Maven*=61 Logan=66. AZLON(10e,34,NTD) or ZOON(11e,26,ADNT) can also be considered.

4. **Logan**(ADEFOTV) plays OFT at h13, 21 pts, *Maven*=61 Logan=87. Of course, you also considered VOTED(5A,27,AF), OVA(H13,21,DEFT), FOVEAL(10b,22,DT), and ADVENT(12c,22,FO).

5. **Maven**(AENNNOT) plays NEON at 5b, 15 pts, *Maven*=76 Logan=87.

6. **Logan**(ACDEEIV) plays DEVIANCE at 12b, 96 pts, *Maven*=76 Logan=183. Another bingo!

7. **Maven**(AHINRTU) plays HURT at 4a, 34 pts, *Maven*=110 Logan=183.

8. **Logan**(DDEEMMN) plays EMENDED at c7, 26 pts, *Maven*=110 Logan=209.

9. **Maven**(ABEINNP) plays IAMB at 8a, 33 pts, *Maven*=143 Logan=209.

10. **Logan**(AILMTTU) plays MATH at a1, 27 pts, *Maven*=143 Logan=236. Strong players also consider UTA(3a,20,ILMT) which scores fewer points but gets rid of the annoying "U".

11. **Maven**(EFGNNPS) plays FEIGN at e10, 18 pts, *Maven*=161 Logan=236. FENS(j9,24,GNP) scores more points, but FEIGN keeps better tiles.

12. **Logan**(AILORTU) plays TUTORIAL at 15h, 77 pts, *Maven*=161 Logan=313. Adam Logan's third bingo!

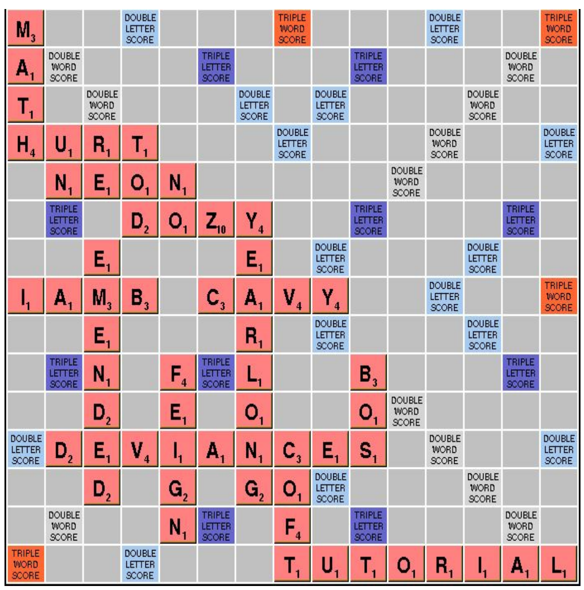

Figure 1: *Maven* plays BOS (j10) scoring 26 points.

13. **Maven**(?ABNOPS) plays BOS at j10, 26 pts, *Maven*=187 Logan=313. See Figure 1. Sheppard considers this to be a "fantastic move" and one of the most difficult moves in the game.

14. **Logan**(IILPRSU) plays PILIS at 15a, 34 pts, *Maven*=187 Logan=347. PILIS, PULIS, PILUS, and PURIS are all good.

15. **Maven**(?AKNPRS) plays SPANKER at k5, 105 pts, *Maven*=292 Logan=347. The only bingo, reviving *Maven*'s chances despite the 160 point deficit.

16. **Logan**(EEEORUS) plays OE at b1, 12 pts, *Maven*=292 Logan=359. The best move, dumping extra vowels.

17. **Maven**(?HJTTWW) plays JAW at 7j, 13 pts, *Maven*=305 Logan=359.

18. **Logan**(AEEGRSU) plays GREASE at m3, 31 pts, *Maven*=305 Logan=390. AGER(L9,24,ESU) also merits consideration.

19. **Maven**(?HRTTWX) plays AX at 6m, 25 pts, *Maven*=330 Logan=390. *Maven*'s second brilliant move, choosing AX over GOX(13G,36) and sacrificing 11 points.

20. **Logan**(EIIILQU) plays LEI at o5, 13 pts, *Maven*=330 Logan=403.

21. **Maven**(?AHRTTW) plays WE at 9b, 10 pts, *Maven*=340 Logan=390.

22. **Logan**(AIIIOQU) plays QUAI at j2, 35 pts, *Maven*=340 Logan=438. A 98 point lead and only a few moves are left in the game. Obviously, it's all over...

23. **Maven**(?AHRTTU) plays MOUTHPART at 1a, 92+8 pts, *Maven*=440 Logan=438. See Figure 2. Wonderful! *Maven* scores exactly 100 points, edging Adam Logan by 2. Sheppard writes that "*Maven* steals the game on the last move. Adam, of course, was stunned, as it seemed that there were no places for bingos left on this board. If I hadn't felt so bad for Adam, who played magnificently, I would have jumped and cheered." This game put *Maven* up by eight games to four, so winning the match was no longer in doubt.

How often do you score 438 points in a game of Scrabble... and lose?

Just in case some of the words used in this game are not part of your everyday vocabulary, here are a few useful definitions (taken from the commercial version of *Maven*):

- Bos: a pal
- Fens: marshes.
- Foveal: a shallow anatomical depression.
- Gox: gaseous oxygen.
- Pilis: a Philippine tree.
- Uta: a type of lizard.
- Zoon: whole product of one fertilized egg.

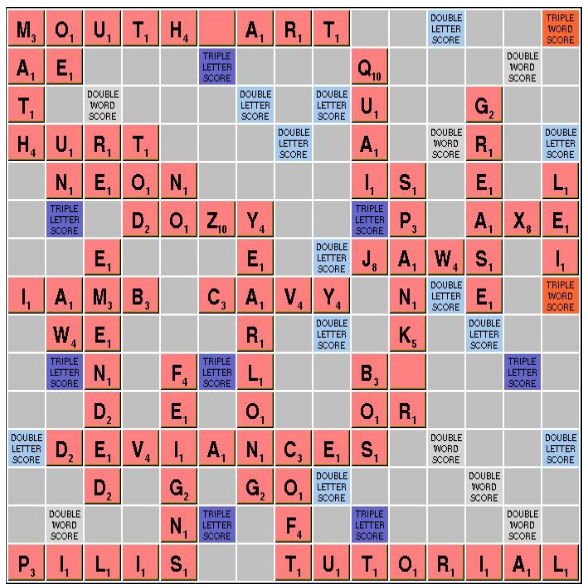

Figure 2: *Maven* — Logan, final position