Paul De Palma: *Dim Sum for the Mind*

Chapter 6

The hard things are easy

My friend, Michael, and I were sitting in Bob Manieri's office a few years ago. Bob had a formal photograph of his four-year-old son on his file cabinet. The boy was seated at a richly brown wooden desk with a shelf of large, expensive-looking and also richly brown books behind him. He was wearing the kind of bib overalls that you often see on toddlers. There was a football on the boy's desk. Michael, a very funny, smart, and sometimes cruel guy, said, "Bob, I didn't know your son was a lawyer." Michael, of course, had gotten it right. The photograph was filled with totemic objects, a football ("Let my son by masculine"), an imposing desk ("Let my son be powerful"), a shelf of very large books ("Let my son be smart"). We could have been Cro-Magnon's in France's Dordogne valley painting bison on cave walls fifteen thousand years ago ("Let those bison, big, beautiful, and edible, come in abundance").

I was thinking about Bob Manieri's office when I opened a text on introductory programming sent to me by a publisher of computer science books the other day. Inside the back cover was a photograph of the author, bearded, middle-aged, hunched in deep thought over a chessboard. What was he trying to tell me? What if, instead, he had given his publisher a picture of himself sitting at the head of a table, surrounded by family and friends, holding a glass of red wine? Or riding a bicycle with his teenage daughter? Or washing the car on a sunny Saturday afternoon? Or sitting at Bob Manieri's son's desk? You get the idea. Bob's photograph was a prayer for his son, suitably encoded. I imagine it buried in a basement for a thousand years, waiting for some future Champollion to unravel its meaning. The writer's photograph encouraged me to adopt his textbook by summoning the cache of mystery and

intelligence that has surrounded chess for two centuries or more ("When I'm not writing complicated books about programming, I play chess, a game that requires intelligence of the same kind needed to solve problems in mathematics and computer science").

Chess and other kinds of adversarial board games have been close to the hearts of computer scientists for fifty years. In 1950, Claude Shannon, a scientist at Bell Labs, and now best-known for his work in information theory, published a paper in *Philosophical Magazine* entitled "Programming a Computer to Play Chess." Remarkably, he specified a framework that is still found in all adversarial computer games. A year later, Alan Turing, developed a chess-playing program before there was a suitable computer on which to run it or a suitable programming language in which to encode it. Turing simulated it in a match with Alick Glennie, a young scientist who would go on to develop an early programming language. As Glennie, a weak chess player, defeated Turing's simulator in twenty-nine moves, computer chess was born. Forty six years later, on May 1997, Gary Kasparov became the first grand master to concede defeat to a computer, IBM's Deep Blue. Much non-technical writing on computer chess is heavy on speculation about machine intelligence, but light on details about how adversarial games are actually built. This is a shame, since the details, though clever, are remarkably straight-forward and, contrary to what most of us think about the pace of invention in computing, have changed little in the fifty years since Shannon first proposed them. In this chapter I will describe several techniques common to all programmed board games. Along the way I'll try to decipher the remarkable hold that chess itself seems to have on the imagination of computer scientists.

Proto-chess probably dates back to the seventh century where a game called *chaturanga* was widely played in northwest India. Though the game was very different from modern chess,

historians regard *chaturanga* as its ancestor because of two unusual features: different pieces had different values and winning depended on capturing a single, special, piece. Muslims brought the game--along with so many other things--to Europe by the tenth century. Within a hundred years or so, chess appeared in the British Isles. We know this through a collection of carved walrus ivory chess pieces, of striking and mysterious beauty, found in the Outer Hebrides in the nineteenth century. Though chess is usually thought to be a stylized version of medieval warfare, in fact, both the relative strength and appearance of the pieces continued to evolve for several hundred years. The first international tournament was held in London in 1851. A German teacher, Karl Anderssen, was the winner and, so, is usually recognized, informally, anyway, as the first world champion.

Chess and its masters have an undeniable charm about them. Their names and stories read like exotic tales from the lives of the saints. We find Ruy Lopez, the sixteenth century Spaniard and inventor of the Classical Defense, Francios Philidor, who could play multiple games blindfolded, the choleric actor and Shakspeare scholar, Howard Staunton, the Louisianan, Paul Morphy, who was driven mad by the game, the Cuban Capablanca, who played as if, in the words of one commentator, "he was the instrument God has chosen to express His will on the chessboard," the Russian Alexander Alexandrovitch Alekhine, whose black eyes and swept-back hair make him a double for Bela Lugosi as Count Dracula, and, of course, more Soviets than Senator McCarthy thought he found in the State Department: Botvinnik, Tal, Petrosian, Spassky, and, finally, the tragic and bitter Gary Kasparov, our century's very own John Henry.

The masters and their quirks blend smoothly with the names of classic moves, evoking, to me at least, a vanished world of dark wood and leather, cigars and brandy: the Alepin Opening, the Alekhine Defense, the Mazukevich Gambit, the Caro-Kann Defense (that was to be

Kasparov's undoing when he played it in his match with Deep Blue), the Goldman Variation, the Gurgenidze Counterattack, the Sicilian Defense, the Queen's Gambit, the Torre Attack, the Leningrad Dutch, to name only a handful. Then there is the endgame, the arena in which the computer has made a genuine contribution to chess and forced a rethinking of the rules for tournament play. These appear mostly as an endless list of book titles: *Pandolfini's Endgame Course*, *The King in the Endgame*, *Secrets of Pawnless Endings*, *Secrets of Rook Endings*, *Mastering the Bishop Pair*, *Essential Endgame positions* and many, many more.

Once we arrive at the many, many books of chess problems and their solutions, this litany begins to sound suspiciously like the history of mathematics. There, as in chess, we find prodigies and eccentric geniuses (Gauss who could calculate before age four, Erdos, the homeless mathematician who lived on coffee and amphetamines, Ramanujan, the self-taught Indian clerk, Galois who was killed in a duel at 21), exotically-named theorems and unsolved problems (Goldbach's Conjecture, Fermat's Last Theorem, The Four Color Map Problem), classic texts (Hardy's *Introduction to the Theory of Numbers*, Russell and Whitehead's *Principia Mathematica*), a comparable list of autistic savants (Trevor Tao in chess, the remarkable twins who generate prime numbers in mathematics), and a predilection for competition, ranking and professional genealogy. With a little digging, you can discover that Alastair Denniston, director of the British Government Code and Cypher School, recruited both mathematicians and chess players to the now famous Bletchley Park where they broke the German Enigma cipher during the Second World War. One of these was the very same Alan Turing whose developed the first chess simulator. The front name for the top-secret Bletchley Park was the Golf Club and Chess Society.

It is not surprising, really, that chess is a game that appeals to mathematicians and computer scientists. All games have a formal element to them. This is what makes them games. Each has a prescribed and inflexible set of arbitrary rules within which you test your skill. The bishop moves diagonally in chess, you have to bounce the ball when you move in basketball, it's better to have four queens than four jacks in poker. But chess—and other adversarial board games like othello, checkers, backgammon and go—has an added appeal for people who take pleasure in formal systems: it is what researchers call a game of perfect information. The board configuration is equally available to both players. There is no bluffing as in poker, no physical gifts as in basketball. The game is entirely internal.

Without pushing the analogy too far, one might argue that mathematics and computing are games of perfect information too. To prove a theorem, one begins with a small set of elements, say the positive integers, a set of operations, say addition and subtraction, various logical constructs like equality and implication, and the set of theorems that have already been proven. The game is to show that some novel state of affairs is implicit in the collection that you already have. But who is the opponent? Well, if you're the Hungarian number theorist, Paul Erdos, the opponent is God whom Erdos calls The Supreme Fascist, because he keeps the best proofs to himself. We in computer science are not so graphic. I've never met anyone who thinks that all programs pre-exist in the mind of some greedy God. But we don't need a God to battle. We have the computer and a set of inflexibly defined rules and relationships. New programs must be implicit in these rules and relationships else our god will let us know that we've failed: it won't recognize our programs or they will run in unpredictable ways.

But what do I mean by "implicit?" I mean that each transformation in a mathematical argument, each division, each addition, each inference is one of many operations that may have

been performed at that step. Further, the arrangement of the symbols after the operation follows the previous arrangement by invoking one of only a handful of rules. But this is exactly what happens in chess (and other board games). Each board configuration is a result of the action of the rules on a previous board configuration. It should come as no surprise to learn that some researchers in artificial intelligence work on automatic theorem provers.

Computer scientists will tell you that they chose to model chess early in the history of computing because board games provide a closed world in which ideas about intelligence might be tested. This may be true. It is surely helpful, however, that the tools required in chess are so similar to those used in mathematics and computing. At the risk of seeming ungenerous to my colleagues, I have noticed that mathematicians and computer scientists admire nothing so much as the kind of skill necessary to do mathematics and computing. It is, in fact, what they refer to when they speak of "intelligence." It should come as no surprise to find early computer scientists hard at work modeling the very skill they most esteem, an observation that I will return to later.

One of the more interesting things about Deep Blue is that, except for it special purpose chess-playing hardware, Turing and Shannon would have understood immediately how it goes about playing the game. This is counter-intuitive. A staple of computing mythology is that software and hardware mutate faster than fruit flies. This is certainly true of capabilities and interfaces. The Web barely existed five years ago and the computer sitting on my desk, rapidly becoming obsolete, would have been tapped for service by NASA a generation ago. Yet TCP/IP, the protocols behind the Web, date back to the seventies and my computer, despite its speed, is still a Turing Machine, an abstract computer that Alan Turing developed in the thirties. Principals are hard to alter, which is why we teach them in computer science programs, even as

students and industry leaders clamor for instruction in Web scripting languages that have the staying power of subatomic particles.

There is some measure of poetic justice in the tenacity of Shannon's observations. Let me explain. Anyone familiar with the contentious history of artificial intelligence will recognize the name, Hubert Dreyfus. This Berkeley philosopher published a RAND corporation report in 1965 criticizing the assumptions behind AI research. The report grew into the seminal *What Computers Can't Do* in 1972. By 1997 it had gone through five editions, launching Dreyfus' second career as the scourge of AI. Dreyfus arguments cannot be easily dismissed. He is probably right when he argues that humans play chess differently than computers (about which more later). And he embarrassed a generation of AI researchers when he exposed their extravagant promises and modest successes[1]. Still, while rereading Dreyfus' book after nearly twenty years, I came upon this passage from a 1958 RAND report by Newell, Shaw, and Simon, authors of some of the earliest work into chess-playing computers:

> In another place, we have predicted that within ten years a computer will discover and prove an important mathematical theorem. On the basis of our experience with the heuristics of logic and chess, we are willing to add the further prediction that only moderate extrapolation is required from the capacities of programs already in existence to achieve the additional problem-solving power needed for such simulation.

And Dreyfus' response:

> Public gullibility and Simon's enthusiasm was such that Newell, Shaw, and Simon's claims concerning their still bugged program were sufficient to launch the chess machine into the realm of scientific mythology.

[1] If you are inclined toward cynicism, you could decode the many early promises as a strategy to secure funding. Even the name itself, artificial intelligence, seems bold, especially since psychologists and educators still don't have a clear notion of what intelligence is. I don't think ill of the pioneers for this indiscretion. We all want to socialize the costs of doing what we like to do. That's why medical researchers leak results to The New York Times and professors claim to be overworked, sports franchise owners demand new stadiums, and defense industry lobbyists argue for missile defense systems.

Simon may have been off in 1957 when he claimed that within ten years a digital computer will be the world's chess champion (it took 45 years to beat a grand master, a big chunk of a human life, but a drop in the 400 year old bucket of modern science), but he was right on the mark about how such a program would play the game. All chess-playing programs, in fact, nearly all adversarial game-playing programs, consist of three parts: a symbolic board description, a move generator, and a position evaluation function. Thus it has been since Shannon set out to solve the problem in 1950.

Representing a chessboard in a programming language is not difficult. You set aside 64 memory cells, work out a scheme so that they can be addressed in row/column fashion, and assign a code and a weight to each piece, say the first letter of its name and an integer indicating its relative importance. The collection of all possible board configurations is called the state space. It is easy to envision the state space for chess represented in what computer scientists call a tree. The root is the game board before any move. Each of the board configurations that could result from first moves is connected to the root with lines that we can call branches. From these new configurations, several moves are possible. The resulting configurations are connected to their parent board states. In simple games, it is possible for a computer construct the entire state space, and search for the path through the tree (a mixed metaphor, I know, but it's standard terminology) until it finds a winning configuration. John Von Neumann[2] and other developers of game theory described this technique in 1944, calling it minimax.

It works quite nicely for simple games. Consider Nim. We start with seven pennies in a single pile. At each move, the player whose turn it is divides a pile into two, each with an

[2] John Von Neumann, incidentally, also invented the general architecture of the computer that is sitting on your desk right now. It's called the Von Neumann Machine in his honor.

uneven number of coins. If a player cannot make a move, the opponent wins. Figure 1 shows the entire state space for Nim.

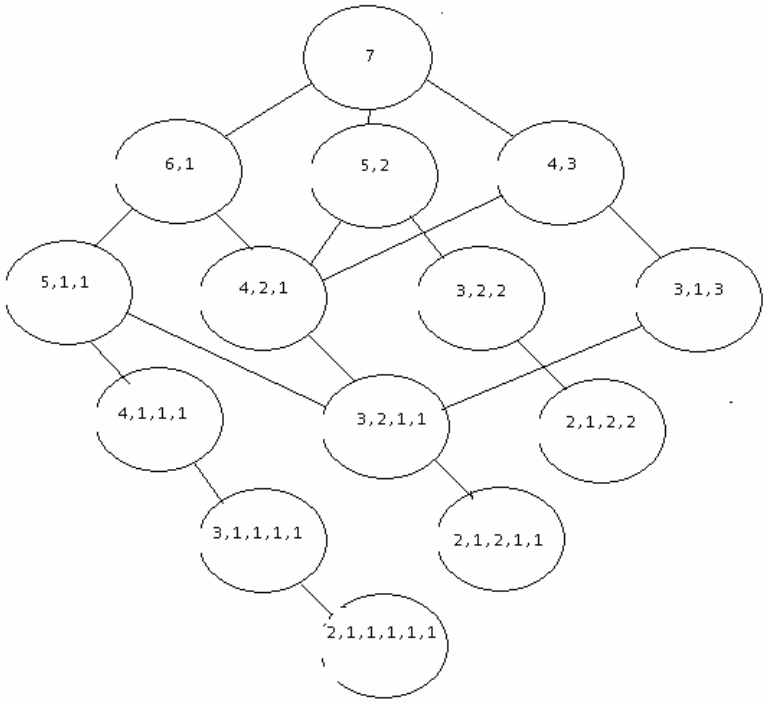

Figure 1

The first player can divide the pile into smaller piles of six and one, five and two, or four and three pennies. If you are the second player and play carefully, you can't lose. To see how, we label each of the levels, or plies, "Max" or "Min." Min wants to minimize her score. Max wants to maximize his. Notice that no moves are possible at the bottommost, or terminal, nodes. If it is Max's turn at a losing state, we label it with a 0, the minimum score If it is Min's turn at a losing state, we label it with a 1, the maximum score. Figure 2 shows Max as the first player. But this is incidental. Min could have gone first, in which case the terminal nodes would have opposite values, 1 where there is now a 0, 0 where there is now a 1.

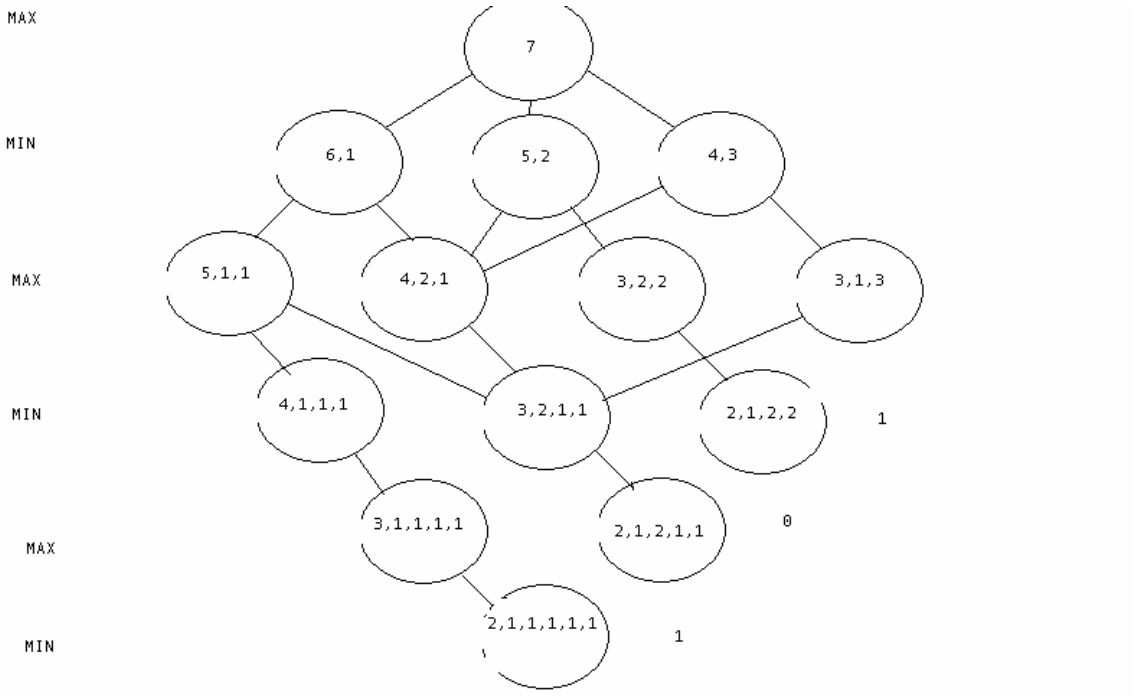
MAX
MIN
MAX
MIN
MAX
MIN
7
6,1
5,2
4,3
5,1,1
4,2,1
3,2,2
3,1,3
4,1,1,1
3,2,1,1
2,1,2,2
1
3,1,1,1,1
2,1,2,1,1
0
2,1,1,1,1,1
1

Figure 2

In theory, either Min or Max could end up in one of the losing states. But our clever Nim-playing computer program can generate the entire state space and knows what the outcome of various choices will be. The trick is to get the information from the terminal node back to the top or root node. Here is the plan. If you are Max, given a choice among moves, you will always choose the move that maximizes your score. But you assume that Min can play as well as you. This is a game of perfect information, after all. Min knows everything that you know. Suppose you are Max, and it is your turn at the nth ply. You will choose a node labeled 1 at the $n + 1^{st}$ ply. If your are Max imagining that you are Min—who said computers can't think—choose a node labeled 0. In short:

For each unlabeled node in the tree:

1. *If it is Max's turn, give it the maximum score of its descendent nodes*
2. *If it is Min's turn, give it the minimum score of its descendent nodes.*

Of course, our program does not know how to score the internal nodes until it descends the tree, scores the terminal nodes, and then works its way back up to the root. There is an equally clever algorithm to accomplish this, called a recursive tree traversal. I'll spare you the details.[3] Here, finally, is the tree completely labeled:

[3] By the way the state space of Nim is not really a tree, but a related structure called a graph. The distinction is not important for our discussion.

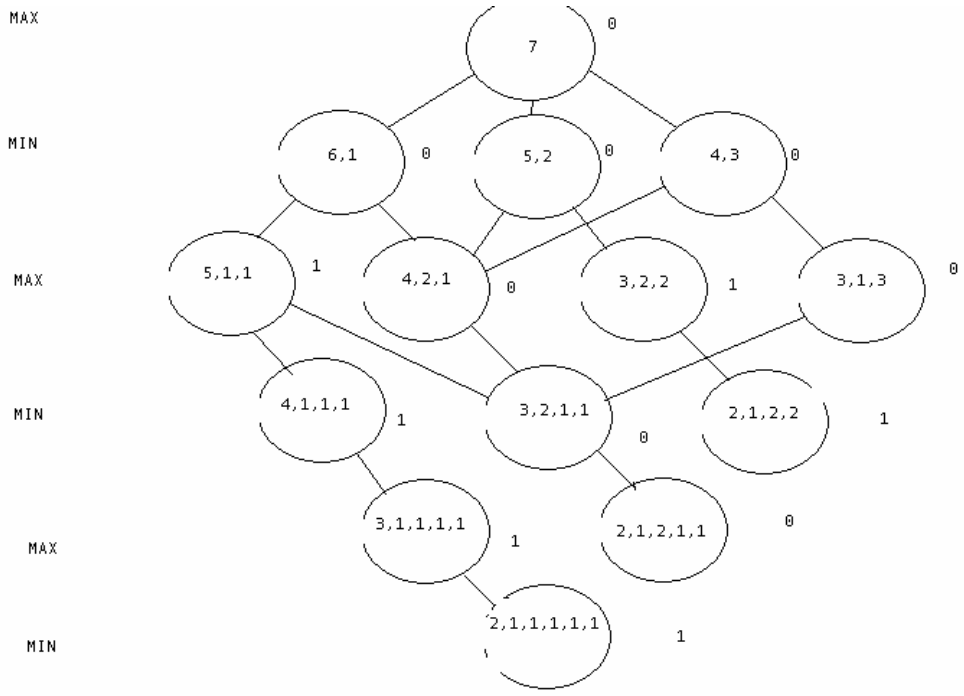
MAX
MIN
MAX
MIN
MAX
MIN
7
0
6,1
0
5,2
0
4,3
0
5,1,1
1
4,2,1
0
3,2,2
1
3,1,3
0
4,1,1,1
1
3,2,1,1
0
2,1,2,2
1
3,1,1,1,1
1
2,1,2,1,1
0
2,1,1,1,1,1
1

Figure 3

If Min goes left instead of right at node (6,1) or right instead of left at node (5,2), she will lose. But she won't do this, because she has foreseen the consequences of bad play and will follow one of the winning paths to the victory at the bottom. Notice that Min can change her mind depending upon what Max does. Suppose she would like to follow the path (6,1),(4,2,1),(3,2,1,1),(2,1,2,1,1), but Max chooses (4,3) on his first move. Min can win the game by choosing (3,1,3) at the next ply.

Many familiar games lend themselves to this kind of exhaustive enumeration and search of the state space. Chess, unfortunately, is not among them. Shannon estimated that at each ply, a player has about 35 options, on average. Now, think of a mountain bike. Each of the three gears on the left has up to six gears on the right for a total of 18 gears (6 + 6 + 6 or 3 X 6). The same idea works in chess. Each of 35 configurations at move n has 35 configurations at move n + 1. So, at two levels we have to examine, on average, 35 X 35 or 35^2 board different board positions. So far so good. Now there are about 30 million seconds in a year, give or take a million or so. Deep Blue can evaluate 2 million board configurations per second or about 60 trillion per year. Here's the bad news. Shannon estimated that there are 10^{120} different chess games. In state space terms, this means that there are 10^{120} different paths through the tree. This is a 1 followed by 120 zeroes, a very big number. AI researchers call this expanding search space, found in nearly every AI problem, combinatorial explosion. Though Deep Blue is incomprehensibly fast, it would need about 10^{106} years to evaluate every position. Since the universe is around ten billion years old and our sun is due to enter its Red Giant phase, engulfing Venus and perhaps earth, a few billion years from now, exhaustive enumeration is not the ticket. Those frightened by the possibilities of digital computing should take comfort. Whether machines think is beside the point. We will never build one that plays a perfect game of chess.

But humans don't play perfect games either. This is the AI part. How do we get the machine to ignore paths that are clearly pointless—some moves are better than others and some are way better—focusing only on those that appear to be a win? Notice the use of the word "appear." As soon as we abandon the hope of optimal play as unattainable, we are in a world unfamiliar to users of computers. What if the word processor you use is suboptimal in the sense that what you type has a probability, however small, of not appearing on the screen? This is not a word processor that most of us would feel very happy about. We might be perfectly comfortable with an assistant who screens our phone calls. Though he may make an occasional mistake, and screen a call from the Nobel Prize committee telling us that we've made the short list, the chances are small and, anyway, to err is human. But what if the screener is a program that sorts through email? Along with the offers to buy discount Viagra or pictures of Totally Wild Wendy ("she shows it all"), it screens a note from the boss asking us to break down an opinion survey by sex and age. We might not be so expansive, nor might the boss. The fact is that we expect our machines to act like, well, machines. The irony is that as computers act more like humans, that is exhibit artificial intelligence, they make mistakes. Not only is this something that we expect from humans and must learn to expect from increasingly sophisticated machines, it is at the heart of certain kinds of problems that we ask computers to solve.

Imagine that you are the operator of an airfreight business. Your planes make a circuit of cities and return their home base. Clearly, the shorter the circuit, the more money your company will make. This is known as the Traveling Salesman (we now say "Salesperson") problem. It is also known that with more than a handful of cities, an optimal solution cannot be found—even by the fastest computers—in reasonable time. Suppose we want to schedule a route of four cities, A, B, C, and D. We begin at A and must return to A. We know the distance between any

two cities. One circuit is A-B-C-D-A. Another is A-C-B-D-A and so forth. If there are N cities in our tour, there are N choices for the first stop, N-1 choices for the second stop, N-2 choices for the third stop etc. Remember the mountain bike. There are N X (N-1) X (N-2) X … X 1 choices in all. This is written N! and called "N factorial." It grows very fast. If our computer can calculate a million possibilities per second, it would take over three years, working round the clock to come up with the optimal tour. You'll be happy, I'm sure, with a good-enough solution. Though this is not, strictly speaking, an AI problem, it does get mired in that slough of despond that AI researchers call combinatorial explosion. All books on artificial intelligence struggle to define the field. I offer my own, with computer chess as the case in point. AI is that area of computer science that develops good-enough solutions to problems where perfection is impossible.

So, how do game playing programs find good-enough solutions? The answer, like so many things in science, is a simple concept dressed-up with a big word. The big word—Greek-derived, of course—is *heuristic*, at bottom, a rule of thumb. Think of Black Jack. Every player knows that as the total value of cards shown on the table grows closer to 21, the chance that the next card dealt will cause the sum to exceed the limit increases. This is a heuristic. We could improve it if we had a way of shifting the probability based on the cards that have been played on other piles. We still wouldn't play perfect Black Jack, but would probably play well enough to get ejected from a Las Vegas casino.

The idea is exactly the same with chess. Not all moves are equally good. A chess-playing computer has to have a way to discriminate among them. The trick is finding a way to discriminate. Before we think about something hard like chess, let's look at a simpler problem. You might remember the 1995 movie, *Die Hard with a Vengeance*. In one tense scene a very

bad guy, Jeremy Irons, presents the two good guys, Bruce Willis and Samuel L. Jackson, with a puzzle. If they fail to solve it, civilization as we know it will come to end. The problem is this. They are given two empty buckets and a source of water. One of the buckets can hold three gallons. The other can hold five gallons. They must deliver exactly four gallons of water in the five-gallon bucket. Lucky for civilization, this is an easy problem, though maybe not easy enough to solve in the two minutes our heroes are given.

If we call the five-gallon bucket X and the three-gallon bucket Y, the following six rules cover all possibilities

1. If X is not full, fill X.
2. If Y is not full, fill Y.
3. If X is not empty, empty X.
4. If Y is not empty, empty Y.
5. If X is not empty, pour the contents of X into Y until Y is full or X is empty
6. If Y is not empty, pour the contents of Y into X until X is full or Y is empty.

It is easy to see this puzzle as a search through a state space. If we start with both buckets empty and run through each of the rules, we find that rules 1 and 2 apply. Now we have two new states. The question is which one we should examine first. That is, which one should we apply the six rules to? It makes sense that we are closer to a solution when the five-gallon bucket holds three gallons than when it holds one gallon.

Suppose we let H be our estimate of the value of a state, a specific amount in the five-gallon bucket and a certain amount in the three-gallon bucket. The lower the value of H, the closer we think it is to the goal. We'll use a fairly simpleminded heuristic: subtract the amount currently in the five-gallon bucket from 4, the goal, and take its absolute value. Suppose the five-gallon bucket is full. The absolute value of 4 –5 is 1. Taking the absolute value allows us to consider the distance to the goal even if we have overshot it. For convenience, we indicate a state as an ordered pair, (X,Y), where X is the amount in the five-gallon bucket and Y is the

amount in the three-gallon bucket. Next, record the level of play as L. The L value for the start state, (0,0) is 0, for each of the 2 states at the next level it is 1 and so on. We define the value of a node, T, to be its H value plus its L value. You can see that if we have two moves of equal goodness, one discovered at level five and the other discovered at level nine, the computer will choose the one discovered at level five. The quicker we find a path through the tree, the greater the chance we have of averting a catastrophe. The total value of a node, T, is just H + L. So that we can return to nodes that may not be promising when we first generated them, but might begin to seem attractive later, we make a list of the nodes we have generated and order it by value. We begin with the start-state on the list. Finally, if you play with this puzzle, you'll find that it will generate states that have already been examined. In computing lingo this is called an endless loop and to be stuck in one is serious business. When your computer freezes up so that neither the keyboard nor the mouse function, it's possible that some combination of keystrokes or mouse clicks threw your program into an endless loop not anticipated by its designers. So that we don't get stuck in one, we'll maintain a second list where we store already examined states. If a newly generated state is present on the second list, don't generate its children. Actually, things are more complicated than this, but in the spirit of good enough, this will do.

The procedure is now quite simple. Examine the node on the list with the lowest T value. By examine, I mean run through the list of rules computing all possible descendents. If two nodes have the same value, examine the leftmost first. After examining a node, put it on the closed list. Stop when you have reached the goal or no further progress is possible. Figure 4 shows the resulting tree. The number inside each node shows the order in which its value was computed. This number corresponds to the entry Node Number in Figure 5, which gives the information I used to create the tree.

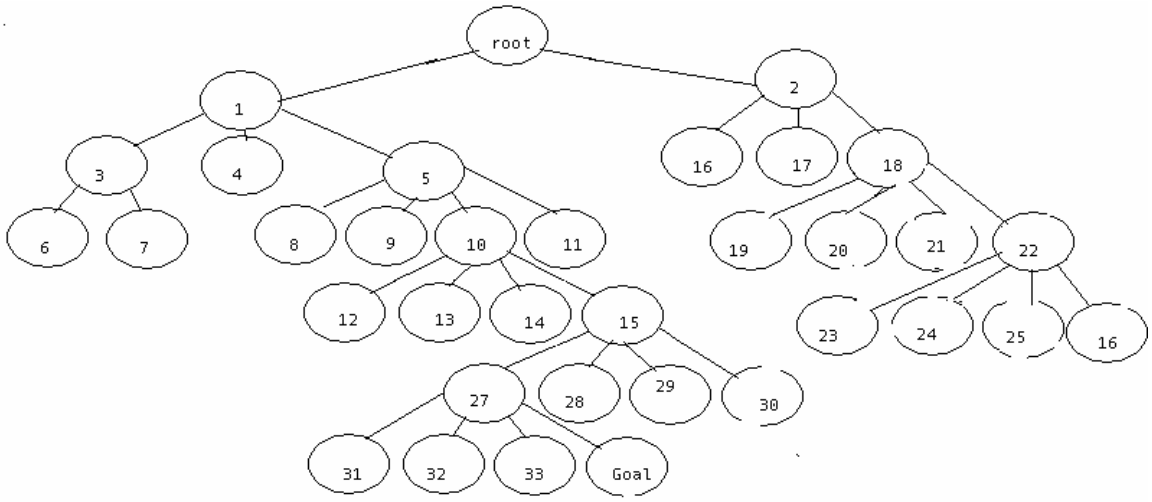
root
1
2
3
4
5
16
17
18
6
7
8
9
10
11
19
20
21
22
12
13
14
15
23
24
25
16
27
28
29
30
31
32
33
Goal

Figure 4

Node Number	Rule Invoked	Five-gal Contents	Four-gal Contents	Value
1	1	5	0	3
2	2	0	3	6
3	2	5	3	4
4	3	0	3	Duplicate
5	5	2	3	5
6	3	0	3	Duplicate
7	4	5	0	Duplicate
8	1	5	3	Duplicate
9	3	0	3	Duplicate
10	4	2	0	6
11	6	5	0	Duplicate
12	1	5	0	Duplicate
13	2	2	3	Duplicate
14	3	0	0	Duplicate
15	5	0	2	Duplicate
16	1	5	3	Duplicate
17	4	0	0	Duplicate
18	2	3	3	4
19	1	5	3	Duplicate
20	3	0	3	Duplicate
21	4	3	0	Duplicate
22	6	5	1	5
23	2	5	3	Duplicate
24	3	0	1	9
25	4	5	0	Duplicate
26	2	3	3	Duplicate
27	1	5	2	7
28	2	0	3	Duplicate
29	4	0	0	Duplicate
30	6	2	0	Duplicate
31	2	5	3	Duplicate
32	3	0	2	Duplicate
33	4	5	0	Duplicate
34	5	4	3	Goal

Figure 5

Here is the first solution that my algorithm found:

1. Start with both buckets empty, Root
2. Fill the five-gallon (Rule 1) → (5,0), node 1
3. Fill the three-gallon from the five-gallon (Rule 5) → (2,3), node 5
4. Empty the three-gallon (Rule 2) → (2,0), node 10
5. Pour the contents of the five-gallon into the three-gallon (Rule 5) → (0,2), node 15
6. Fill the five-gallon (Rule 1) → (5,2), node 27
7. Fill the three gallon from the five gallon (Rule 5) → (4,3), Goal

Generating the entire tree from left to right is called a breadth-first search. It is what we would have done if we considered only the depth of the tree when computing the value of a node. In the case of chess, it's what would happen if we considered every move at every level. Breadth-first search will always find the shortest path to the goal if there is a goal. The downside is that it might take a long time. If you have nothing to do for a few hours, do a breadth-first of the tree until you discover a solution. I hope you can see that my heuristic, however, primitive, saved a lot of effort. On the other hand, in a different setting, it could miss the shortest path to a goal or even miss the goal altogether. This might happen if a heuristic made a particular path look better than it actually is. It is a guess, after all. On the other hand, if you do find the goal with your heuristic, you'll find it more quickly than you would have without it. In games where we can't search the entire tree, this is precisely the distinction between an optimal solution and a good-enough solution that I made earlier. As it happens, a heuristic is not always obvious. This is the tricky part. As the game gets more complicated, the difference between a hastily conceived heuristic and a good one, is the exactly the difference between a beginning player and an expert.

When we use a heuristic to eliminate nodes that do not look promising, we are pruning the tree. In complex games like chess, even after we do a heuristic evaluation down to a predetermined depth, the tree is still substantial. It turns out that further pruning is often possible. Newell, Shaw, and Simon, the same team that suffered under Dreyfus's lash, developed an extraordinarily clever technique called alpha-beta pruning, though the technique is said to have been first described by John McCarthy, the same John McCarthy who, in a stroke of public relations brilliance, coined the term Artificial Intelligence in 1956. Using alpha-beta we can often further prune a tree after minimax has assigned values to nodes. There are some ideas in computer science that are so elegant, and so obvious once explained, that had I dreamt them up, I would spend the rest of my days a happy man. Alpha-beta pruning is one of them.

The idea is that whether you are Min or Max you should go as deeply into the search space as time permits. Suppose that you can look ahead N plies. The alpha-beta technique lets you establish a relationship between the subtree rooted at ply N-2 and the subtree rooted at ply N-1. If certain simple conditions hold, the subtree rooted at node N-1 can be eliminated without having been explored. Here is a simple example:

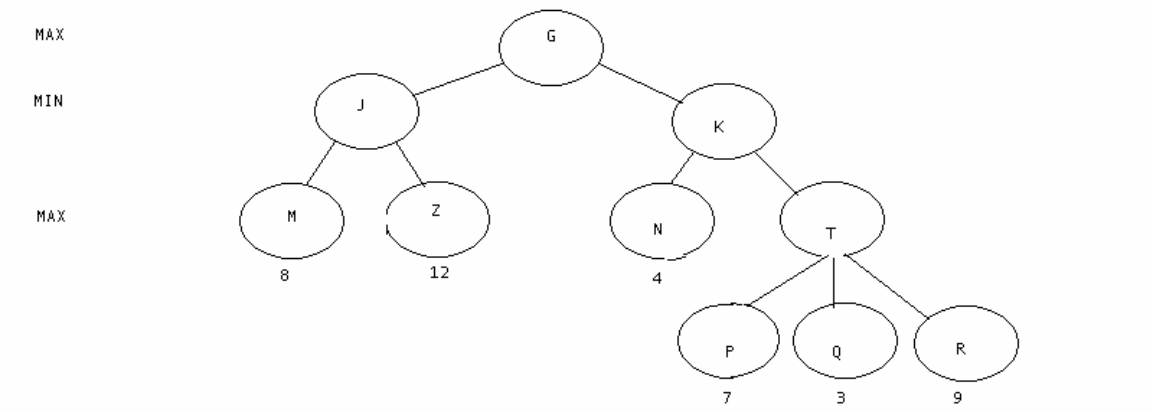

FIGURE 6

Suppose the terminal nodes have actual values as labeled. It is Max's turn at node G. Max, recall, is tries to maximize his values. Min is tries to minimize his. One of the things that each player must do is use an orderly method to examine the nodes in the tree. We have already discussed the breadth-first technique which examines every node at level n before going on to level n + 1. To do an alpha-beta prune, the players must examine the tree depth-first fashion. This means descending left as far as possible before going right. Without going into the details, this means that Max compares the values at nodes m and z. Notice that at this point Max is trying to imagine what Min might do. So if Max chooses j over k, Min will certainly choose m over z. So, Max assigns an 8 to node j. This is called the alpha value. This is the best that Max can do if he goes left. The left subtree of g having been examined, Max is ready to go right. After going to k, he goes left to find that node n has a value 4. This is called the beta value and herein lies the magic. Notice that if Max choose node k, Min would choose between n or t. If t were less then n, Min would certainly choose it. So the worst that Min can do if max chooses k is 4. Put another way, if Max chooses j Min is guaranteed 8. If Max chooses k, Min can do no worse than 4. Clearly, Max is better off with j and he knows this without even having examined t and its children. I call this the Max principle: Search can be stopped below any Min node whose beta is less than or equal to the alpha of its Max ancestor. In figure 6, this means that we can assign a value of 8 to node g. You can convince yourself of the benefits to the alpha beta prune by doing a conventional minimax labeling of the unlabeled nodes in Figure 6. You will find that g retains its value of 8. So in this tree of ten nodes, alpha-beta let us label g correctly after having examined only 6 nodes, a 40% savings. Not surprisingly, there is a mirror-image pruning procedure from Min's point of view as well. It is necessary to invoke this if either Min were node g or if state space were larger than the one shown. This is just a detail. The idea to

walk away with is that in a situation where exhaustive search is impossible, alpha-beta lets us search more deeply than the straight minimax technique. In principle, a chess-playing computer will use the same technique. In fact, Deep Blue, using the alpha-beta algorithm and searching around 40 billion positions for each move was able to increase search speed by a factor of 40 billion, depending on the board configuration.

In 1965, the Russian mathematician, Alexander Kronrod said that chess is the *drosophilia* of artificial intelligence. But, as McCarthy cleverly points out, if genetics had developed like computer chess, geneticists would have spent their time breeding racing fruit flies. This would have resulted in some science but mostly very fast insects. He is clearly irritated with the attention given to the ever-increasing speed of chess-playing programs, as if what emerges from the laboratories of electrical engineers has anything at all to say about artificial intelligence. Nevertheless, Deep Blue, like all chess-playing programs, has to choose among board configurations. As it happens, it does this quite cleverly. The first step is to assign a value to each piece. A pawn gets a one, the queen a nine, and the king, though slower than his wife, less graceful than his bishop, more cumbersome than his knight, receives the highest possible value since his loss means the loss of the game. But things are more complicated than this. Frederick Freidel, Gary Kasparov's computer advisor—Kasparov sparred with a machine called Fritz to train for Deep Blue—tells a story about conversations with grand masters Walter Browne and Max Euwe. Freidel asked Browne about a board configuration in which each side had the same pieces and controlled the same amount of space. Though any amateur would have declared the position a draw, Browne told Freidel that White controlled more space. When Freidel pointed out that each side controlled exactly the same number of squares, Browne answered, "Oh, these squares don't count. They aren't important." This might have remained an untested

idiosyncratic judgement, had not Freidel presented the same problem to another grand master, Max Euwe, who answered in exactly the same way. This gets to the heart of the matter for a programmer building a chess-playing program. How does one formalize the pattern that two grand masters recognized? A baby step is to recognize that the value of a piece is based on its type as well as on the amount of territory it controls. Do this--a significant task--and your machine is at the amateur level. Deep Blue does this and more. Since the outcome of the game depends on the safety of the king, Deep Blue looks at the positions of both kings and assigns values to them. It then tries to put its king on a safer square and drive the opponent to move his king to a more vulnerable square.

Along with these standard procedures, Deep Blue has taken a step in the direction of acquiring the skill Frederick Freidel observed in the grand masters. It appears to play, at least at certain points, positional rather than tactical chess. One plays positional chess when there are no clear pieces to be captured or squares to be controlled, when what is needed, for example, is an assessment of the opponent's pawn positions so that moves can be made from which to launch more tactical and strategic plans. It is the kind of play in which grand masters excel. Oddly, when one plays positional chess, it can look as if he is changing plans between moves. In fact, more than one observer has noticed that chess-playing computers have a kind of ad hoc quality to them, exactly what you would expect from a machine that bases its next move on supposing its component plays exactly as it does.

But Deep Blue is not an ordinary chess-playing computer. Any good chess-playing computer will have a database of board configurations and recommended moves to be played early in the game. Chess players call this the "opening book." Deep Blue's was relatively small, consisting of several thousand positions. To compensate for this, its developers introduced an

algorithm that derives moves based on a Grand master game database. So, before doing the kind of search described earlier, Deep Blue first determines if a move is in the opening book. If it is, it plays it. If not, it looks for the board configuration in the extended book. Now, the extended book will offer alternative moves. Deep Blue chooses among them using criteria like the number of times a move has been played by a Grandmaster, the strength of the Grandmaster who played the move, and the commentary on the move in the position database. Based on these and other factors, Deep Blue can award a particular move half the value of a pawn of that move has been successfully played many times by strong players in the past. This had serious consequences for the Kasparov match. In the second game, Deep Blue's first nine moves were from a classic Ruy Lopez opening. The next eight were derived from the extended book. Deep Blue won in forty-five moves. The AI pioneer, Herbert Simon, goes so far as to attribute the strength of its play in the Kasparov match to Deep Blue's ability to notice patterns of pieces, apparently adjusting the direction of its search to the pattern of a given board configuration. This is much more like the way humans play chess than blindly searching 40 billion positions in each three minute move. Emanuel Lasker, a world chess champion from the turn of the last century said it with the characteristic bravado of a Grandmaster: when asked how many moves he examined when evaluating a position, he is said to have replied, "Only one. But it's always the best move."

Still Deep Blue can search those 40 billion positions. No matter how clever its tutors, without the capacity to run a very deep search, it could not have beaten Kasparov. So, how does this happen. For starters, it does not happen on the computer you use to browse the Net nor on the one I am using to write this chapter. Deep Blue requires significant special purpose hardware and, more to the point, significant institutional support. The institution is the most venerable of computer manufacturers, IBM. It seems fair to ask why a company whose reputation is based on

reliability and—let's be frank—a certain dullness would want to build a game-playing computer. As it happens, chess is not the first game that IBM supported. As far back as the fifties, clever engineers were building game-playing machines on IBM time. It seems that IBM management was ambivalent about this effort, since it carried overtones of malevolent machines thinking for themselves and IBM was in the business of, well, business machines. It appears that IBM still has misgivings. On its Web page it includes a short essay by the head of IBM's RS/6000 division who says that the real issue in building Deep Blue was not to win at chess but to "help pharmaceutical companies develop innovative drug therapies, auto makers design cars and petroleum companies explore for oil buried deep under the ocean. It also helps forecast the weather, clean up toxic waste sites and safeguard the U.S. nuclear stockpile." In other words, forget that visionary stuff about artificial intelligence, we wear blue suits. We're IBM.

Whatever its reasons, IBM has been developing a chess-playing machines since at least the late 80's, under the leadership of IBM's parallel systems designer Feng-hsiung Hsu. At bottom, Deep Blue relies upon an array of chess chips, each capable of examining 2 to 2.5 million positions per second per chip. Though it was possible to design a faster chip, the designers opted instead to integrate the chips into an IBM RS/6000 supercomputer. Now the RS/6000 super computer is really a collection of RS/6000 workstations connected through a high-speed network. For Deep Blue, there were 30 workstations, each with 16 chess chips for a total of 480 chess processors. If the work done by a single chess chip were performed on a general-purpose computer it would require about 40,000 instructions to examine a single position. With a little arithmetic, we can see that Deep Blue was executing the equivalent of 100 billion instructions per second. Interestingly, Deep Blue's search uses a combination of hardware and software. Suppose it searched 12 levels deep. One of the workstations, acting as

the controller for the entire system, would search the first four levels in software. By level 5 the number of board configurations to be examined increases enormously. All 30 work stations search these configurations in parallel for 4 more plies. Since the number of board configurations increases exponentially with the plies, the number of positions to be examined by level 9 is staggering. Now all 480 chess chips offer their services for the remainder of the search.

The question to ask at this point is what does Gary Kasparov do when it's his turn. The answer, of course, is that no one really knows, including Gary Kasparov. In the forties, the Dutch psychologist and expert chess player, Adriann de Groot, studied the game. His most surprising result is that masters and amateurs consider about the same number of alternatives, spend about the same time on each alternative, and look just about as far into the future. The difference between the levels of play appears due to the amount of chess knowledge the master can summon. Somehow, the master can see things in a given board configuration that we mortals pass over. Isn't this exactly what we mean by expertise, the ability to see a pattern—in the configuration of pieces on a board, the symbols of a theorem, the notes on a score—among objects that look unremarkable to the rest of us. And we are back where we started: how do we formalize in a computer program whatever it is that the expert knows? The answer, again, is that no one really knows, but the designers of Deep Blue, in preparation for the Kasparov match are closer than they were a decade ago. They will be there when Deep Blue knows more and calculates less. This will happen one day, exactly because no matter how subtle a position might appear, an expert can explain its dynamics.

This is another way of saying that the world of chess is a closed system. In fact, with its tokens and well-defined operations on those tokens, it resembles nothing so much as

mathematics itself, as I've already mentioned. This brings me back to our bearded computer scientist pondering a chessboard. If there is a place where the words "genius" and "brilliant," are used as frequently as in mathematics, it has to be chess, probably for the same reason: both are closed systems where there is a clear distinction between novice and expert. The history of mathematics is awash in sentiments like "the brilliance of his mathematical discoveries before he was twenty years old," "despite his obvious mathematical genius," (from Motz and Weaver's *The Story of Mathematics*) and with book titles like *A Journey through Genius* (by William Dunham). Similarly the grand master Reuben Fine's history of great chess games tells us about the "most brilliant master of all time," "his most brilliant game," "as has always been the case with chess geniuses" and so on. It is no accident that a textbook writer has himself photographed at a chessboard. The capacity to ascribe significance to objects is deeply human. Spend a couple of weeks touring the great monuments of the Italian Renaissance and you'll find St. Lawrence portrayed with the grate on which the Romans roasted him, St. Peter Martyr with a hatchet in his head, St. Agatha, virtue intact, holding a plate with her severed breasts, and, in that land where unmarried men are said to live with their mothers in their forties, more paintings of Madonna and Child than there are American flags at a Republican rally. These are the objects with which saints achieved sainthood, offered to us as models of piety. So also, the chess-playing computer scientist displays his brilliance, like a male peacock in full regalia, with his chessboard. When the inventors of the infant discipline, artificial intelligence, fifty years ago, set out to develop thinking machines, they chose to model chess, a game requiring a kind of intelligence close to their own. Now any textbook on artificial intelligence will tell its readers that researchers have chosen to work on problems like chess—or much simpler games—because they provide an arena in which the rules are well-understood so that theories of machine intelligence can be thoroughly

modeled. What they do not say is that the kind of problem solving necessary for expert chess—or mathematics—is just the kind of intelligence that researchers in AI admire most in the all the world. After all, weren't the smart kids in high school the ones who studied math and physics? And isn't the world divided equally by the Educational Testing Service between mathematical and verbal aptitude? The great irony of fifty years of research in artificial intelligence is that the kinds of skills, like expert chess-playing, that very few of us possess can be modeled computationally, whereas those abilities that any normal human develops effortlessly, language is the best example, have so far eluded us. The hard things are easy (relatively speaking) and the easy things are hard.

References and Further Reading

Atkinson, George. *Chess and Machine Intuition*. Norwood, NJ; 1993. Of the many books on artificial intelligence and of the many on chess, this short book comes closest to capturing what is distinctive about chess-playing computers. Along the way, it offers fascinating mini-histories of the same of the important figures.

Bregman, Mark. "With deep blue technology, we all win**."** Retrieved 9/6/02 from: http://researchweb.watson.ibm.com/deepblue.
The manager of the division that developed Deep Blue at IBM assures us that IBM is not interested in chess for its own sake.

Campbell, Murray. "Knowledge discovery in Deep Blue." *Communications of the ACM*, November (42:11) 65 - 67; 1999.
Describes how knowledge of chess was encoded in Deep Blue.

"Chess." The New Encyclopedia Britannica. 1998 ed.
An excellent overview of chess and its history

Dreyfus, Hubert. *What Computers Still Can't Do: A Critique of Artificial Reason*. Cambridge: MIT; 1997.
Though many people have criticized AI over the years, no one has done it as consistently and convincingly as Hubert Dreyfus.

Fine, Reuben. *The World's Great Chess Games.* NY: Crown; 1951.
A personal tour of the chess greats by one of the masters.

Hsu, Feng-hsiung. "IBM's Deep Blue chess grandmaster chips." *IEEE Micro* March-April: 70-81; 2000.
A technical description of the architecture of the Deep Blue hardware by its designer.

Hoffman, Paul. *The man who loved only numbers: the story of Paul Erdos and the search for mathematical truth*. NY: Hyperion; 1998.
Mathematical biography, it turns out, is its own subgenre. Though one might doubt that the biographical details of people who spend most of their waking hours in mental contemplation would be compelling, Hoffman proves the doubters wrong. Mathematicians include a disproportionate share of odd and interesting characters. Erdos is one of the oddest and most interesting.

Kanigel, Robert. *The man who knew infinity: A life of the genius Ramanujan*. NY: Charles Scribner's and Sons; 1991.
Ramanujan was a self-taught Indian mathematician. His relationship with the D.H. Hardy in the early years of the twentieth century is touching. It made me long for the days before

standardized testing and academic degrees would make it impossible for someone like Ramanujan to emerge from obscurity.

McCarthy, John. "AI as sport." *Science* June 6; 1997.
A critical review of the effort to build a computer that can play first-rate chess. John McCarthy is said to have invented the term "artificial intelligence" at the 1956 Dartmouth Conference where the field was christened.

McCorduck, Pamela. *Machines who think.* San Francisco: W.H. Freeman and Company; 1979.
An early, in-house history of AI as enthusiastic as Dreyfus is sceptical. I include it because McCorduck reprints and interview with Arthur Samuels, an early checkers programmer, who recalls that his employer was never very enthusiastic about his work.

Newborn, Monroe. *Kasparov vs. Deep Blue: computer chess comes of age*. NY: Springer; 1997.
Though published before the Kasparov's defeat, this book is a compendium of chess games and chess-playing programs that led up to Deep Blue.

New York Times. Spring and summer; 1997.
The *New York Times* published many pieces on the Kasparov / Deep Blue match, including a daily summary of the games. George Johnson, Bruce Weber and Robert McFadden wrote some of the more interesting pieces.

Waldrop, M. "How the chess was won." *Technology Review,* August/September, 33-36;1997.
A nice interview with Deep Blue's principal designer, Feng-Hsiung Hsu.