11

Computer Chess – A Case Study on the CDC 6600

D. N. L. Levy
The Computing Service
University of Glasgow

Abstract

This paper is essentially a state-of-the-art survey of chess programming as typified by the most recent program to appear on the scene. In order to be able to view the situation objectively we feel that it would be useful to preface this with a historical review of the development of ideas in this twenty-year-old field. By considering the most important ideas and techniques that are employed in the (currently) best program available, we hope to convince the reader that progress has been very slow despite the multiplicity of programs (and their associated literature) which have appeared since 1950.

HISTORICAL REVIEW

The most important paper that has appeared on the subject of computer chess is one written by Claude Shannon in 1948 and published two years later (Shannon 1950). Shannon's paper does not describe an actual program, but offers many suggestions for those who are interested in writing one. In this respect Shannon's paper may be compared with one by Jack Good which was also full of sound ideas which could well be included in a successful chess program (Good 1967).

Shannon stressed the importance of having a good evaluation function. The features which he considers necessary for inclusion in the evaluation function included material, mobility, five aspects of pawn-structure, four of the positions of pieces, and four of commitments of pieces, attacks and options. He appreciated that such an evaluation function should be used only in the middle-game, and that different principles applied to the opening and endgame phases of chess. He suggested that the values of the coefficients of the function should be determined by 'some experimental procedure', and the fact that this statement has never been followed in practice is very surprising.

Chess programmers seem to arrive at the coefficients for their evaluation functions by (hopefully) intelligent guesswork, and then alter them in the light of experience. No one has even attempted to use Samuel's method (Samuel 1957) for allowing the program to improve its coefficients itself, and this illustrates what we feel is the 'head in the sand' approach of many workers in this field. This approach consists of the following three-stage plan for writing a chess program, a plan which, unfortunately, seems to have become universally acceptable: (1) write a program that plays legal chess, guessing at the coefficients of the evaluation function; (2) play some games against the program, adjusting the coefficients in whatever way seems indicated and making any other alterations that appear to be necessary or promising; (3) *if* tired of chess programming *then* give up *else goto* stage 2. The two most obvious stages that are missing from the above plan are: (−1) read the literature to avoid making the same errors as previous workers and to take advantage of their good ideas; and (0) THINK about what chess programming is all about, preferably including some discussion with strong chess players.

Another most important point that was made in Shannon's paper is the inapplicability of evaluation functions to nonquiescent positions. The logic behind this is trivially obvious: if analysis of a particular variation is being terminated in the middle of an exchanging or forcing sequence of moves, the most important features in the evaluation function will be meaningless. It is of no value, for example, to be a queen ahead if one's opponent has the move and he can recapture the queen to level the odds. Shannon suggested that minimax be used to search the game tree, but that instead of searching to some fixed depth under all circumstances, there should be a bound beyond which the program would search only when it was analyzing a forcing sequence. He also put forward the idea of forward pruning, that is, only analyzing the variations that superficially seem to have some point. His other ideas included the use of a backing store for 'memorizing' openings, changing the style of play of the program by altering the search parameters or the coefficients of the evaluation function, and incorporating some sort of learning mechanisms along lines roughly similar to Samuel's.

Shortly after the appearance of Shannon's paper, Turing published his thoughts on the problem (Turing 1953). Most of his ideas involved assigning arbitrary weights to various features in the evaluation function. His simulated program was the first experimental venture in the history of the subject. It exhibited poor and aimless play.

One of the features which is of considerable importance in chess is mobility – the ability to choose one's move from a large rather than a small set. Turing's measure of mobility consisted of summing the square root of the number of moves that each piece (queen, rook, knight, or bishop) could make, though he failed to give the reasoning behind this idea.

A group working at the Los Alamos Scientific Laboratory in New Mexico

(Kister *et al.* 1957) wrote a program to play chess on a six-by-six board (omitting the bishops). Apart from introducing the idea of weighting moves into the opponent's territory more than those into one's own (thus producing more aggressive play on the part of the program – rather a mixed blessing), their work added nothing to the art. Their program played a weak game, equivalent to an amateur who has had about 20 games experience. At the end of their paper the authors express the opinion that 'a machine will be built in the near future which could be coded to beat a strong player'. If their prophecy has come true, no one has yet managed to prove it.

A program was written for MIT's IBM 704 during 1957 and 1958 (Bernstein *et al.* 1958). This was the first program to arouse much publicity and the first one to play acceptable chess. A few important heuristics were used in a (reasonably successful) attempt to simulate human behaviour. Moves were generated in an order which tended to reflect their likely usefulness (human players, naturally enough, consider 'obvious' moves first). The generation of moves was performed in response to various questions put to 'decision routines', the most important decision routine ('Is the king in check?') was entered first and an affirmative answer resulted in the generation of moves which either moved the king, captured the offending piece, or interposed a piece. Less important routines asked questions such as 'Is any of our material under threat?' and 'Is castling possible?', and naturally enough the least important routines were 'Can a pawn be moved?' and 'Can a piece be moved?'.

The program selected the seven best moves in each position for further examination, the merit of a position being assessed by an evaluation function with only four features: mobility, material, area control, and king defence. Analysis was carried out to a depth of four half-moves, but the size of the tree was kept down by a routine which allowed examination of a move only if it appeared to produce an increase in score from the previous position. Restriction of the search in this way in conjunction with the forward pruning mechanism resulted in a program which played as well as a 'passable amateur' and which took only two minutes per move on average.

The authors realized the importance of using different strategies (in their case different decision routines) to play the endgame, and they also appreciated the need for some learning mechanism to improve both position evaluation and the ordering of moves within the decision routines. Their work on chess programming, unfortunately, ceased before they were able to implement either of these ideas.

Newell, Shaw, and Simon started their work on chess programming in 1955 (Newell *et al.* 1958). They introduced the goal→subgoal approach to problem solving in their work on GPS and the ideas carried over to their work on chess. Their program contained a routine which determined what 'state' the program was in (that is, what kind of position had been reached). A second routine decided which of a list of possible goals were applicable to

that state. (Compare the thought process of a chess master who asks the question 'What sort of plan is likely to succeed in a position of this kind?') Lastly there was a routine which generated the moves that were associated with the goal(s) which had been selected. This whole process is very similar, in essence, to the decision-routine approach of Bernstein and his colleagues with the important difference that, whereas Bernstein's routines were in a rigid order, Newell, Shaw, and Simon's subgoals were ordered by chess heuristics for each different position. Their work contained one other notable conceptual breakthrough, namely, the introduction of the alpha-beta algorithm. They made no particular comment about their own method of tree searching, and one wonders whether perhaps they did not realize that this was to become such an important technique in heuristic programming.

From 1958 until 1967 there was a recession of interest in chess programming, the only exception being a program written by Alan Kotok for his bachelor's thesis at MIT (Kotok 1962). Kotok's program did not add anything new to the art, but it did serve as a basis for the Stanford program which played a match against a Soviet program in 1966–67. The fact that the Soviet program won by a score of 3–1 (two wins and two draws), despite being run on a much slower machine, suggests that it must have contained some useful original ideas. The only reference to work actually completed in the USSR is contained in a paper which appeared in 1966 (Adelson-Velsky *et al.* 1966). Their paper is full of generalities and does not, in general, contribute to the current state of the art, but the authors make the very interesting suggestion of speeding up program running times by introducing hardware move generation. This could clearly make a difference of several orders of magnitude in the time taken to generate all the legal moves in a position, but it would result in a special-purpose machine and therefore detract from the aim of writing a good chess program for a general-purpose digital computer.

In 1967 Richard Greenblatt's now famous program appeared (Greenblatt *et al.* 1967). About fifty chess heuristics were implemented, some of which were used to vary the width of search in order to ensure that exchanging sequences and variations involving checks were examined until (hopefully) they reached quiescence. Other heuristics were used to order the moves so that optimum use might be made of the alpha-beta algorithm. A few 'book' openings were stored, largely to prevent the program from falling into a standard opening trap.

The performance of Greenblatt's program represented a marked improvement over Bernstein's, but this was due to programming skill rather than to any conceptual breakthrough. The program also owed a great deal to the improvement of hardware over the previous decade, and this leads us to believe that, unless a concerted effort is made to write a really first-rate chess program, the state of the art will only progress as a function of hardware speeds.

Another notable contribution to the field, which appeared in 1967, is the theoretical paper by Good. The number of ideas expressed therein is so vast that it is not feasible to list them here, but we feel that it is worth commenting on the fact that in the three years since Good's 'Five-year plan' was published there has been no attempt made to implement it. This leads us to the conclusion that chess programmers are perhaps too obsessed with their own ideas and not sufficiently interested in testing those of others.

In 1968 John Scott, while still a schoolboy, wrote a program for Lancaster University's ICL 1909 (Scott 1969). The program played at below the Greenblatt level and did not contribute anything new.

Summary

We summarize the most important developments of the first two decades of chess programming in table 1.

Table 1

Year	Author(s)	Significance of work	Comments
1950	Shannon	Contained far more good ideas than any subsequent paper up to 1967.	No more than half of his suggestions have been implemented.
1958	Bernstein *et al.*	The first program with any real ability.	Demonstrated the power of simulating human thought-processes.
1967	Greenblatt *et al.*	The first program capable of a reasonable performance in a tournament.	Owed most of its success to improved hardware and efficient coding.
1967	Good	Contributed a wealth of new ideas.	None have been implemented.

Conclusions

Many people have worked on chess programming during the past twenty years, too many in fact for us to include mention of all of them here. Most of these have adopted the three-stage plan which is doomed to failure, but a few have had some original thoughts on the subject and have contributed to the art. Unless there is a great conceptual breakthrough in the field, we feel that the only way in which a really strong program could be written is for a

concerted effort to be made along the lines of Good's five-year plan. Programmers and chessplayers working in conjunction might have a chance of producing an outstanding program, but we do not feel that either group, working on its own, is likely to have too much success.

THE STATE OF THE ART

The program to which we now turn our attention was written for the CDC 6000 series operating systems by two CDC employees (Larry Atkins and David Slate) who developed the program at Northwestern University, Evanston, Illinois, with the help of Keith Gorland, who is a student there. This part of our paper really belongs to the programmers because the information that we have on their program is entirely derived from the comprehensive printout that is produced when the program plays a game. The program is, in our opinion, certainly the strongest to have been written, that is to say, we ourselves experienced more difficulty in beating it than we did with either Greenblatt's program or Scott's. What is the fairest method of comparing the strengths of programs written for machines of different sizes and speeds is a subject for separate discussion.

The program and working space occupy about 30,000 words (octal). Input is through the console, and the current board position as well as the program's reply moves are displayed on one graphical display. On an adjacent display each player has a clock which is set initially, under the standard operating conditions at ten minutes, but which can be altered either before or during a game. The depth and width of search also have standard settings, though these too are open to change, as is a parameter which determines whether the program should play in a tactical or a strategic style.

When a player types his move the piece that he moves appears to glow for a second or so and then moves to its new square where it once again glows before taking up its permanent stance. The input/output facilities are so good that it is really a pleasure to play the program just viewing the display – a chess board and set of men are unnecessary. Running under the standard settings the depth of look-ahead is three half-moves. This can be varied up to a maximum of seven, and we shall discuss the optimal settings later. The width of search at each level is another crucial feature that can be altered at the player's discretion. If the widths are $w_1, w_2, \ldots, w_j (j \leqslant 7)$, then the program will examine a maximum of w_i moves at level i. In addition, all checks and captures are examined if an option called 'combo' is initialized. The program thus performs forward pruning with reservations. The depth of search also has a flexibility option which causes seemingly-inferior moves at level one to be examined to a depth of one extra half-move. Needless to say, this results in a certain amount of time-wasting, but on the other hand it prevents the exclusion of moves which, while being superficially poor, are actually worthy of examination.

There is a rote-learning procedure which may be called as another option. When 'rote' is on, the program stores each player move and the corresponding board position in its library. When the program reaches that position in a future game, it automatically makes the same move as the player did, and it is thus possible to teach the program openings and standard endgame wins. An option called 'analyze' monitors the progress of the program when playing against itself or against another player, and uses large changes in its estimated position score to assign blame or credit to the previous four moves.

The programmers have not yet produced any documentation which describes the heuristics that they use, but the program printout lists some thirty features that are employed in their evaluation function, and we feel that this is the key to the program's relatively strong play. The coefficients of the function may be changed at will and this, in conjunction with the options mentioned above, makes the program very easy to modify. A strong chess player who was allowed sufficient time to perform several case studies such as the one which we are about to discuss would probably be able to obtain great insight into the workings of the program's 'mind' and teach it to improve its play much as one might teach a child. The one feature which the program will need before it can be taught to play chess at a really high standard is the ability to form simple concepts and to make generalizations. This is an essential partner to the rote-learning procedure, and once the problem has been overcome a few months' training by a grandmaster could raise the program's standard of play to an unprecedented extent.

THE CASE STUDY

The following game was played between the University of London's CDC 6600 and a human player whose rating on the United States Chess Federation Scale is about 2050 (the British Champion would be about 2550 and the World Champion 2700; the author's rating is 2380). The program was searching to a depth of five half-moves with the widths of search set at 4, 4, 4, 4, 1. Having constant width settings down to the penultimate level of search seems to be a good idea as it compares well with the chess master approach of examining only a small number of moves from every position. The basic settings (that is, those in operation when the player has not made any adjustments before commencing play) for a depth three search are 8, 16, 1 and we feel that 12, 12, 1 would probably produce some improvement in play without increasing execution time. Bernstein's program was set at 7, 7, 7, 7 though we do not know whether these settings were chosen by trial-and-error experiments or in some other way.

The program took about one minute per move and its opponent about five to ten seconds. Much further experimentation will be necessary before it is possible to estimate the strength of the program with any accuracy, but we feel that its grading would be around 1750. The program would therefore be about the 500th best player in Britain. It will be interesting to see how the

program fares in the Islington Open Chess Championship in late December.

S. Reuben *v.* CDC 6600

1. P–Q4 N–KB3

The program does not have a store of openings at its disposal and this choice of move was therefore made on the same basis as when playing a middlegame position. This is a fairly serious fault which is being remedied at the moment by the inclusion (in the next version of the program) of a number of standard opening variations. Nevertheless it will still be essential to introduce a different set of heuristics for the openings – this point was made by Shannon twenty years ago and it has never been put into practice.

The program's choice is in fact the most common move (in reply to 1. P–Q4) that is seen in master chess!

2. P–QB4 N–B3?!

In contrast this move is justifiably rare as it allows white to obtain a significant advantage in space, for example, by 3. P–Q5. It would seem that the program puts slightly too much emphasis on the development of pieces and not enough on controlling important central squares with the pawns. 2. ... P–KN3 and 2. ... P–K3 are the accepted moves here.

3. N–QB3 P–K3!

The only satisfactory move. 3. ... P–Q4 could be answered by 4. P×P KN×P 5. P–K4 N×N 6. P×N, and 3. ... P–Q3 by 4. P–Q5 N–K4 5. P–B4 N–N3 (not 5. ... N×BP?? 6. Q–R4ch winning the knight). 6. P–K4. In each case White would have complete control of the centre. However, in view of the fact that the program's second move gave White an opportunity similar to the second of these variations, we cannot assume that the program played 3. ... P–K3 for the right reason.

Despite having played the best move here, Black still has not recovered from the effect of his second move and 4. P–K4 would now give White a considerable advantage.

4. B–N5 B–N5

Once again this active move is best, and White's reply almost entirely dissipates his advantage. 5. P–K4 would be very strong.

5. P–K3 B×Nch?!

The program knows that doubled and isolated pawns are disadvantageous and at once takes the opportunity of implementing these rules by landing White with weaknesses on the queen's side. In such positions there is no need to make this capture until White has wasted a move by P–QR3, and development by 5. ... P–QN3 would have been more appropriate. Such subtleties however are only learned after many years experience and at the time that this game was played the program was less than one year old.

6. P×B P–KR3

Forceful play, probably prompted by the desire to relieve the pin on the knight which is in fact of no danger to Black. It is rather impressive that in an

evaluation function employing far more than the usual number of features the programmers have a numerical model which seems capable of frequently playing the correct move, albeit for the wrong reason.

7. B–R4 P–KN4
8. B–N3 P–Q3!

A solid move which prepares for the possibility of ... P–K4. A weaker player might have tried 8. ... N–K5 9. B–Q3 N×B (not 9. ... N×QBP 10. Q–B2 winning a piece) but the program's evaluation function correctly realizes that after 10. RP×N White's pressure on the black KRP would outweigh the slight disadvantage of the doubled KNPs.

9. B–Q3

9. P–KR4 would be bad on account of 9. ... N–K5 10. P×P N×B 11. P×N Q×P, but 9. N–B3 would leave Black in some difficulty because 10. P–KR4 would be a dangerous positional threat: for example, (9. N–B3) N–K5 10. B–Q3 N×B 11. RP×N, and against almost anything else 10. P–KR4 creates serious weaknesses in Black's king's side.

Now Black has the advantage.

9. ... P–K4!
10. P–KR4

10. P–Q5 would permit Black to bring his queen's knight to the unassailable square QB4 via QN1 and Q2, and 10. P×P N×P leaves White's queen's side pawns waiting to be picked up like ripe plums.

White hopes to take advantage of Black's weakened king side, but the program does not fall for 10. ... NP×P 11. B×RP (when Black has no satisfactory defence against 12. Q–B3; for example, 11. ... P–KR4 12. Q–B3 R–R3 13. B–N5 and White wins a piece), or 10. ... P–N5 11. P–R5 (when 12. B–R4 would also be unpleasant).

10. ... B–N5
11. Q–R4

11. P–B3 B–R4 12 P×P R P×P would be quite satisfactory for Black.

11. ... K–K2!

The only move. White was threatening 12. P–Q5 (winning a knight) as well as 12. RP×P (winning two pawns), and 11. ... N–Q2, the only other move that prevents loss of material, fails to 12. P–Q5 which leaves Black severely cramped.

12. P–Q5 N–QN1
13. P–B5 QN–Q2
14. P×Pch P×P
15. Q–N4 Q–B2!

Simultaneous attack and defence. Naturally 15 ... N×P? is not possible because of 16. Q×B.

16. P–QB4 N–B4
17. B–B2 N–R3

By a series of simple moves the program has built up an extremely strong position. White must go into great contortions if he is going to defend his QBP. For example, 18. Q–R4 KR–B1 19. B–N3 N–K5 with a great game for Black.

18. Q–B3??

A gross blunder after which only the length of the game is in doubt. As we have mentioned White was taking only a few seconds over each move – normally a player of his standard would not overlook such a simple trick.

18. ... N×P

To as good a tactician as the program this move is trivially obvious.

19. Q–Q2 N–N3
20. B–N3 N×P
21. Q–B3 N–R4
22. Q×Qch

Two pawns down with no compensation. White has what would normally be considered a totally lost game. Under such circumstances the most practical approach would be to keep queens on the board and to hope to be able to perpetrate some 'swindle'. The human player had heard that 'computers can't play endgames' and therefore considered the exchange of queens to be his best chance.

22. ... N×Q
23. R–B1 N–N4

23. ... N–K3 seems to be a more logical alternative, putting a piece in the centre rather than near the edge of the board. However the program would see this move as one that unnecessarily squandered mobility since by 24. B×N White could exchange a bishop which had six moves at its disposal for a knight that had seven. This brings out a fault in the commonly-used method of position evaluation which seems only to have been appreciated by Greenblatt. In such an important feature as material, the difference in material between the two sides $(M_w - M_b)$ is not such a good measure as $(M_w - M_b) \times$ (greater of M_w and M_b)/(smaller of M_w and M_b) which implies the heuristic 'swap off pieces when you are ahead in material'.

24. N–K2 N×B
25. P×N KR–QB1
26. K–Q2 B–B4

This time the program's emphasis on mobility serves it well by redeploying the bishop on a better diagonal.

27. P×P P×P
28. R–R5 R×R
29. N×R P–B3
30. P–B3 P–Q4

To prevent 31. P–K4

31. P–B4 R–QB1
32. N–Q3

Not 32. P×NP R–B7ch 33. K–Q1 N–B6ch, etc.

32.	...	B–N3
33.	R–R6	NP×P
34.	R×B	K–B2
35.	R–R6	K–N2

The program's last two moves indicate that it is still playing as though it were in a middlegame and trying to keep the king as near to the edge of the board as possible.

36.	R–R3	P×B
37.	R×Pch	K–R3
38.	N–N4	N–B6

Threatening 39. ... N–K5ch.

39.	R–B3	K–N2
40.	R–B1	P–R4!
41.	N–Q3	N–K5ch
42.	K–Q1	R–B6

With his last three moves Black has assured himself of winning another pawn.

43.	N–B1	R×KP
44.	K–B2	P–Q5
45.	R–Q1	P–N4
46.	R–Q3	

White is still sceptical about the program's endgame ability.

46.	...	R×R
47.	N×R	K–B2
48.	K–N2	N–B6
49.	N–B5	P–...
50.	K–B2	P–K6
51.	N–N7	P–K7
52.	K–Q2	P–R5
53.	P×P	P×P
54.	N–Q6ch	K–K3
55.	N–B4	P–B4

White is fast running out of moves.

56.	N–R3	K–K4

So that after

57.	N–B4ch	K–Q4
58.	N–R3	P–B5

White's mobility is further reduced.

59.	N–B2	K–K5
60.	N–R3	K–Q4
61.	N–B2	P–B6??

Black can clearly afford to give away a pawn at this stage but there is no good reason why he should have done so. The program is avoiding repeating

the position although it could have done so (for the second time) before playing ... K–B5.

62.	P×P	K–B5
63.	P–B4	

If 63. N–R3ch K–N6 64. N–B2 N–N8ch wins.

63.	...	P–Q6
64.	N–K3ch	K–N6
65.	N–N2	P–R6
66.	K×P	P–R7
67.	P–B5	P–R8=Q
68.	P–B6	Q–KB8
69.	Resigns	

Comments

Considerable advances have been made in the standard of play in the middlegame and these can be attributed to the relatively large number of features in the evaluation function. In a tactical struggle such as this the program is completely at home and it would be interesting to see how it fared against an opponent with a solid positional style.

The program is still lacking in the openings but this can be remedied by the storing of the opening literature. What is likely to present much more of a problem is the endgame. To be able to win when two or three pawns ahead is simple, but at master level the endings that occur most frequently are those in which one side is only one pawn ahead or in which material is level and the advantage is qualitative. Many of these endings require extremely deep and detailed analysis and to program them would be a formidable task.

THE FUTURE

The author is so pessimistic about future progress that he has made a £750 bet with Professors McCarthy, Papert, and Michie that there will not be a program that can beat him in a match by August 1978. The basis for this bet is not any lack of confidence in the ability of workers in this field but of their lack of application. It has taken twenty years to produce a program that can play reasonably well in the middlegame. How many more will it be before the endgame is programmed to the same standard? The answer to this question is complicated by the fact that no one has even attempted the problem yet.

Two ex-world chess champions (both computer experts) were recently asked their opinions of the author's bet. Dr M. M. Botvinnik said 'I feel very sorry for your money'. Professor M. Euwe held entirely the opposite viewpoint. Only time can tell.

Acknowledgement

I should like to thank the University of London Computer Centre for giving time on its CDC 6600, and Mike Baylis for his help in arranging the game reported here.

REFERENCES

Adelson-Velsky, G.M. Arlasarov, V. L., & Uskov, A. G. (1966) Programme playing chess. Symposium on theory and computing methods in the upper mantle problem.

Bernstein, A., Roberts, M. de. V., Arbuckle, T., & Belsky, M. A., (1958) A chess-playing program for the IBM 704 computer. *Proc. Western Jt comput. Conf.*, 157–9.

Good, I.J. (1967) A five-year plan for automatic chess. *Machine Intelligence 2*, pp. 89–118 (eds Dale, E. & Michie, D.). Edinburgh: Edinburgh University Press.

Greenblatt, R., Eastlake, D. & Crocker, S. (1967) The Greenblatt Chess Program. *Proc. Fall Jt comput. Conference*, 801–10.

Kister, J., Stein, P., Ulam, S., Walden, W. & Wells, M. (1957) Experiments in chess. *J. Ass. comput. Mach.*, **4**, 174–7.

Kotok, A. (1962) A chess playing program for the IBM 7090. Bachelor's thesis. Department of Electrical Engineering, MIT.

Newell, A., Shaw, J. C. & Simon, H. A. (1958) Chess-playing programs and the problem of complexity. *IBM J. Res. Dev.*, **2**, 320–35.

Samuel, A.L. (1959) Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, **3**, 210–29.

Scott, J.J. (1969). A chess-playing program. *Machine Intelligence 4*, pp. 255–65 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Shannon, C.E. (1950) Programming a computer for playing chess. *Phil. Mag.*, **41**, 256–75.

Turing, A.M. (1953) Digital computers applied to games. *Faster than thought*, pp. 286–95 (ed. Bowden, B.V.). London: Pitman.