Programming ENTER: Christopher Strachey's Draughts Program

David Link

This article details some problems – and some solutions – encountered when resurrecting a program for the game of draughts from 1951 on an emulator of the Ferranti Mark I.

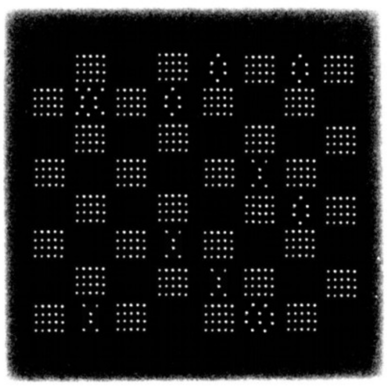
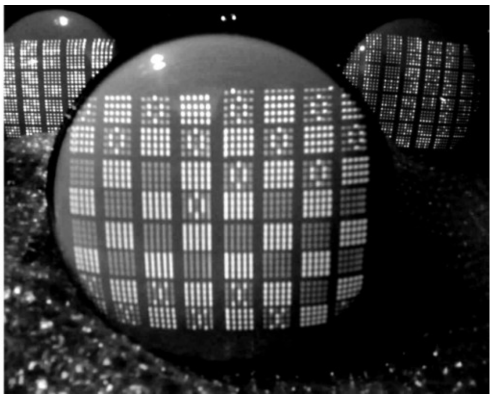
The Ferranti Mark I was the industrial version of the Manchester Mark I, whose prototype, the Manchester "Baby" (SSEM), performed its first calculation on 21st June 1948. The algorithm here described was one of the earliest complex applications authored on the pioneer computer that did not only serve system testing purposes. Christopher Strachey, an outsider to the Manchester computer laboratory and a school teacher, had developed the software in his spare time. Martin Campbell-Kelly writes, relying on the oral histories from lab personnel:

"Strachey sent his programme [draughts] for punching beforehand. The programme was about 20 pages long (over a thousand instructions), and the naiveté of a first-time user attempting a programme of such length caused not a little amusement among the programmers in the laboratory. Anyway, the day came and Strachey loaded his programme into the Mark I. After a couple of errors were fixed, the programme ran straight through and finished by playing *God Save the King* on the hooter (loudspeaker). On that day Strachey acquired a formidable reputation as a programmer that he never lost."

The material relating to the draughts program has been preserved in the Strachey papers in the Bodleian Library, Oxford. In it, there are found approximately five versions of an algorithm that is about 20 pages long, pencilled on the usual Manchester coding sheets. There are also printouts of sample games Strachey played against the machine at the time, which were recorded on the teleprinter. Dates on the papers indicate the software was mainly developed in June and July 1952. A first, undated version was probably written prior to May 1951. (In a letter dated 15th May 1951, Strachey wrote to Turing: "I have completed my first effort at the Draughts" and he was obviously talking about the Manchester Mark I. At this point, the algorithm already had "input and output arrangements"). In February 1951, the Ferranti Mark I had been installed in Manchester University. Strachey gave a lecture about Draughts at the ACM conference in Toronto in September 1952, which was published in the Proceedings.

Game Machine User Experience

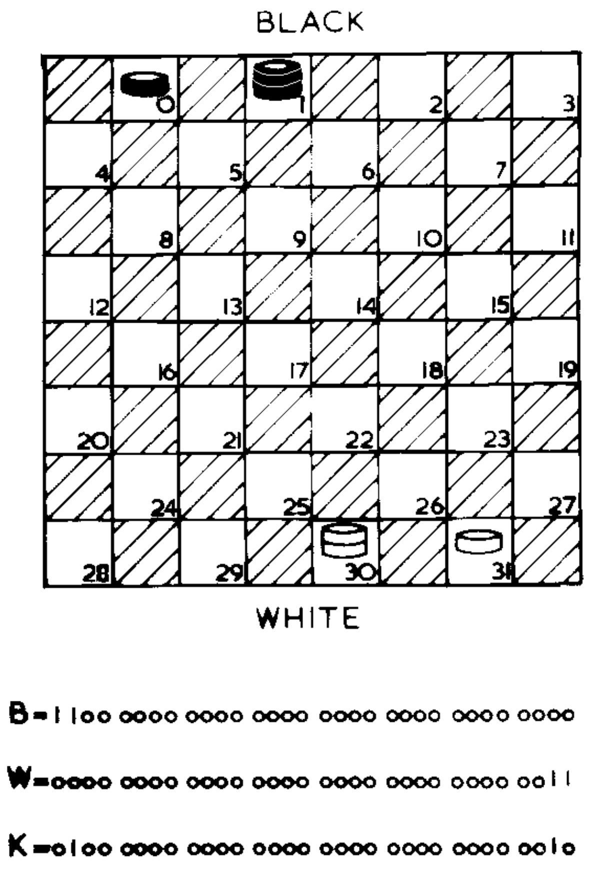
When the software started, it asked the user to **PLEASE READ THE INSTRUCTION CARD** on the teleprinter. He would then hit a key labelled "KAC" on the console to signal he had done so. The algorithm asked him to spin a coin and claimed either heads or tails. The user let the program know via a switch and KAC if it had won or not to determine who had the right to start the game. Then human and machine made moves alternately, the latter by printing them on the teletype, the former by setting the hand-switches on the console and hitting KAC. The complete game was printed out, and two consecutive situations could always be inspected in parallel graphically on cathode ray tubes 3 and 5, which were part of the working memory of the machine. The software very probably constitutes the first usage of a graphical display in a computer program.

The draughts board as shown by the storage CRT of the Ferranti Mark I, and a modern recreation by the author

Strachey had coded an additional "preview feature": after the user had announced his move by setting it up on the switches, the machine showed the resulting position on cathode ray tube 3. If he then answered NO by composing **///T** on the console (bit 19 on), the algorithm reverted to the previous situation on the board and he could try something else. If the user input wrong information, the machine became increasingly angry, until it uttered: **I REFUSE TO WASTE ANY MORE TIME. GO AND PLAY WITH A HUMAN BEING/**. (The slash was probably used as an exclamation mark, which was missing in the teleprinter code.) A similar routine existed if the opponent took too long to reply. Strachey had apparently become fascinated with the slightly obscene theatrical effect of a machine making human-like statements and showing "emotion". His next software was an inversion of this rather strict, impatient character, a program for the composition of love letters. Draughts already contains the complete "rhetoric" that is needed for it algorithmically, including the selection of pre-fabricated text based on random numbers.

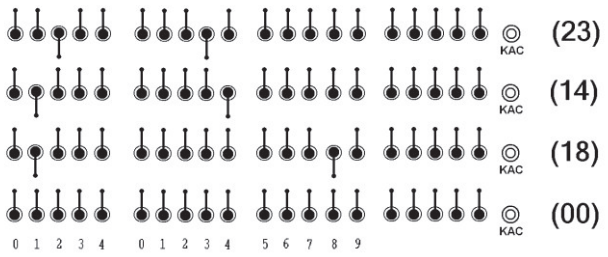
Coding a Game


BLACK
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
WHITE
B=1100 0000 0000 0000 0000 0000 0000 0000
W=0000 0000 0000 0000 0000 0000 0000 0011
K=0100 0000 0000 0000 0000 0000 0000 0010

For the coding of the situation on the board, the white fields had been numbered from 0 to 31, and three 32-bit variables (memory locations) named B, W, K respectively expressed the positions of black pieces, white pieces and kings of both colours by setting the corresponding bit = 1.

A move sequence, on the other hand, consisted of two values in the same range, the fields from which and to which the piece was displaced, after which the position of a captured piece could follow, for example 23-14, with the opponent hit on 18. The program also mastered multiple captures correctly. For setting up moves on the hand switches, Strachey employed an intuitive system rather close to decimal, where the first five bits indicated the tens (0 to 3), and the second and third the units (0-9) of the position number.

In this way, sequences such as “23-14 (18)” could be expressed as:



(To end the move sequence and return control to the machine, the user had to hit KAC with nothing set. The first bit in each group signifies 0, the second 1, and so forth.)

The strategy implemented in the game algorithm was a heuristic one, so one could claim draughts was the first heuristic program too. Strachey wrote that the difficulty of “the machine to look ahead for a number of moves and choose its move by a valuation scheme” was of great theoretical interest and presented a typical example of a “large logical programme”. Other games were less challenging, because “most of them use the complete mathematical theory so that the outcome … is no longer uncertain”. His program calculated the next move by looking ahead for a certain number of steps without being able to overview the complete game. By not trying to exhaust the endless number of combinatorial possibilities, he “succeeded in making a programme …, which will play a complete game of Draughts at a reasonable speed.” In fact, this is not true: There is no code to control the end game, to detect it is over and to announce a winner. To write a program that could handle the rather complex task of playing draughts must have been sensational at the time.

The central element in the heuristics of the algorithm was the evaluation function for future positions. In it, the machine calculated all possible moves up to a certain depth and summed up the material left on the board resulting from each, counting three for a king and one for a normal piece. Theoretically, i.e. from the perspective of storage space, the algorithm could look ahead three operations on each side (with depth = 6), but in fact, due to the much more pressing limits on time, it was in most cases only anticipating three in total (depth = 3). Quite typically, as actual program performance can be very different from the planned one, the strategy had the serious flaw that the machine started to behave suicidally: As a result of the valuation scheme, it sacrificed all its pieces when the user was about to king. Strachey met this by adopting two different depths of search in such a way that in case one of the last two moves had been a capture, the machine calculated on. After that, it kept looking ahead until the second depth value was reached. (In the run on 10.7.1952, this value (b) had been 1, with a (normal search) = 5.)

Strachey had separated the strategic core of the algorithm from the service functions and commented: “It is rather typical of a logical programme: that this organising routine is in fact longer than the game-playing routine proper.” The latter was called DRAUGHTSB or DR/B and consisted of eight pages (in the version dated 10.7.1952), while for the service part (DRAUGHTSC) occupied another ten sheets, with four containing auxiliary functions. So, 18 or 22 pages in total, depending on the method of counting — incredibly long for the time.

Resurrecting Draughts

In the course of software reconstruction, usually parts start to work while others still malfunction and ultimately lead to a crash of the program one tries to resurrect. One technique here is to follow the algorithm through and to find the exact point where it starts to go wrong. This is usually slightly earlier in the executed code than the final crash. (It is astonishing how long programs can sometimes run on completely wrong grounds.)

When the exact position of the aberration is found, this particular place in the code can be investigated and probably be fixed, provided the situation is not too complex. The software will then continue to execute, until it encounters another crash point, or ideally run through to the end, in which case the reconstruction succeeded. This technique of debugging already existed in the 1950s and there were dedicated “check sheets” to trace or log a program at runtime, i.e. to record the memory locations that changed in the sequence of the operations of the algorithm.

In one such situation in the beginning of the resurrection of Draughts, the program was waiting for some time, and then went to a “hoot stop”. This was the symbolic equivalent of a crash, by which the software signalled that something had gone fundamentally wrong.

Upon closer inspection, the algorithm was stuck in the following lines (see note below for an explanation of the notation):

```
1 – T/: Accumulator (A) = 147456
2 – TN: A = A–1
3 – /M: go to line 2 if A >= 0
4 – /I: switch M and L, the left and the right side of (A)
5 – /H: continue execution of program if A >= 0
6 – /T: go to hoot stop
```

In line 1, the 80-bit accumulator is set to a rather high number, 147456, by copying it from address VK in the working memory. It then counts this quantity down by subtracting the contents of address E: from it, which holds 1. This location is part of two pages of values that are kept in memory permanently, PERM. The third line is a conditional statement: If the accumulator is greater than 0, go to operation 2, where the number is again decremented. At one point, the value there will change from **//////// ////////** to **££££££££ ££££££££**, that is, from 0 to -1. Since a command takes 1.2 milliseconds to complete on the average, this will happen after approximately 5.9 minutes. The algorithm then continues in line 4. The operation here exchanges the left (L) and the right (M) 40 bits of the accumulator. Since it is set to all 1s, this produces the same number, **££££££££ ££££££££**, which is -1. In line 5, the algorithm jumps to what is obviously the continuation of the program, if and only if the quantity in the accumulator is positive! Otherwise it enters the already-mentioned hoot stop – an endless loop with no break condition, which consists of the following two lines:

```
1 –  /V: hoot
2 –  /P: go to 1
```

In modern notation, the algorithm we just discussed could be rewritten in the following way:

```
int i = 147456;
while(i >= 0) i--;
switchMandL(i);
if(i < 0) hootStop();
else continue program execution
```

This code seemed to make no sense at all! To understand it, it is useful to consider how signed numbers were represented in the Manchester Mark I. Generally, these were 40 or 80 "binary digits", written with the most significant bit to the right. The handbook specified: "On the plus-minus convention the most significant digit is used to represent the sign." To find out if the number in the 80-bit accumulator was positive, it was sufficient to have a look at bit [79]: When it was 1, the number was negative. The machine automatically copied the value of this bit to the A-sign flip flop, and in case of an A-conditional statement, it consulted the data there. So again: How could the switching of the two sides of an accumulator full of 1s result in the 79th bit becoming zero? Apparently, the algorithm expected something that could never happen, an impossible event. Formulated differently, it was waiting for a miracle. (In very much the same way, the tautology **while(true)**, which encloses the run loop in the core of most programmes, can only be broken in the improbable event that truth is no longer truth.

In the operating instructions, Strachey wrote that the "machine gives a 'pip-pip' signal when it requires attention. It should always be restarted by operating KAC after it [the machine] has been set appropriately." He went on to give examples of what the computer would say and in which way to react to it. The KAC key was one of the several clearing switches the Manchester Mark I inherited from the "Baby" prototype and its function was to empty the accumulator. But would hitting KAC not lead to the same situation as counting it down until it reached zero? In both cases, the accumulator would first become all zeroes, = 0, and then all ones, = -1, when it was decremented in line 2. It was impossible to see any reason why the switching of the two parts would make the number positive. And yet, it was quite obvious that the code in question could do exactly this: tell the difference between counting down and hitting KAC.

Analogies in Logical Design

The solution to the riddle was that Strachey relied in his programming on the logical design of the Mark I, its hardware properties. I failed to make sense out of the code fragment for a rather long time, because I was looking at the machine on a purely symbolic level, where signs were transformed into other signs instructed by signs. The emulator was only an implementation of the Mark I's operation codes and its effects on the contents of the stores. In this mode of thought, pushing a button was treated like a command, and more importantly, like a synchronous one. There was no difference between KAC and the operation code `T:`, which also cleared the accumulator.

In writing, meaning is conveyed by material elements, the words and letters. In the same way, the data and operations in computers are represented by certain real systems with suitable properties, by a physical analogy. The function to clear the accumulator is implemented in certain electronic components, a Williams tube by the name of "A", in a way that follows the logic of this device. Since something is stored here if it is refreshed, it is sufficient to prevent recirculation to delete the data.

But it is not only the spatial physical analogy that counts, but also the temporal aspects of this simulation of thought processes. On the most basic level, computers move in cycles, which are subdivided in a number of phases in which certain predefined elementary actions take place. In the Manchester Mark I, there were seven of them: SCAN1 to SCAN3, and ACTION1 to ACTION4. The timing with respect to these also determined if an operation was synchronous or asynchronous. In the logical design picture of the machine we are interested in here, it is important in which phase certain parts of commands are executed.

Programming ENTER

In comparison to the activities of the algorithm when it counts down the accumulator from 147456, what happens differently on the logical design level when the user hits KAC? The 80 bits of A are set to 0 and the software subtracts 1 from it, making it negative, the A-sign flip flop is set and the program breaks from the first loop. What is important here is that the sign of A is identified after the arithmetic, but before the number re-circulated returns to the accumulator. Upon leaving the loop, the number (-1) is not re-circulated and hence A is empty again. When the algorithm switches M and L in a later cycle, the A-sign flip flop is clear and the program jumps to its continuation, not to the loop stop. The rather elaborate sequence thus simply detects if the KAC key has been pressed. In that case, the software jumps into the following code fragment:

```
1 –  /J: M += //// ///E
2 –  /H: go to 1 if A >= 0
```

First, a number is added to the right part of the accumulator, equivalent to adding 1 into its 75th bit. Then, if the number is not negative, the procedure is repeated. Again, it seems quite impossible that by adding a positive quantity, the result can become negative. Obviously, the user is still holding down KAC when these statements are reached, which prevents recirculation, leaving the accumulator empty. Once the key is released, it starts to increment and at the 16th addition this carries over into the sign bit. The algorithm jumps to its continuation. The code thus detects the release of KAC and waits if it stably stays in this position to prevent accidental bouncing of contacts to disturb user interaction. With the fragments described above, it formed a detection sequence for the typing (press / release) of the key.

“Phew! that was a good exercise”, wrote Christopher P. Burton after mostly he had found out what the mysterious fragments meant. So the code was actually not waiting for the impossible. Strachey had simply constructed in software what would today be called an ENTER key. He needed it because of the way in which the user should communicate with the software: He set up on the console hand-switches an answer like the next move to be played and signalled he had composed it by depressing ENTER. Interestingly, no key to “send” the carefully composed information to the machine existed on the console of the Mark I. But luckily enough, with some ingenuity, it could be programmed.

The Manchester Mark I and its Notation

The Draughts program ran on the Ferranti version of the Manchester Mark I and Strachey used the notation established by Turing in the programming manual. The machine was based on a 20-bit word, and 20-bit numbers (and also instructions) were specified as four 5-bit elements, each element taking the name of the teleprinter code equivalent to the 5-bit value. Thus binary '00000' was expressed as '**/**' and '10000' (least significant on the left) as 'E'. The written form of numbers and instruction was quite opaque unless one was very familiar with all 32 of the possible teleprinter codes. The 5-bit value '11111' was written as '**£**'.

Most instructions contained a function (operation) number and a store address. The function number was six bits long so could be expressed as two teleprinter characters, the first of which was always '**/**' or '**T**' ('00000' or '00001'). The instructions relevant to this article have the following meanings:

/H	Jump direct if accumulator >= 0	**/T**	Jump direct unconditionally
/I	Exchange most and least significant halves of accumulator	**/V**	Hoot (sound the loudspeaker)
/J	Add contents of a store location to most significant half of accumulator	**T/**	Load accumulator with contents of a store location
/M	Jump relatively if accumulator is >= 0	**TN**	Subtract contents of a store location from accumulator
/P	Jump relatively unconditionally	**T:**	Clear the accumulator

The Accumulator is 80 bits long, containing four 20-bit words. The most significant bit (bit 79) is the sign bit, 0 meaning that the number in the accumulator is zero or positive, and 1 meaning the number is negative.

The author is indebted to Chris Burton for solving this, and other, enigmas. David Link would be very happy to hear from all readers who remember Strachey's draughts program. He would also be extremely grateful for any hint on the whereabouts of other Mark I software, especially in the areas of meteorology, nuclear physics and chess. He can be contacted at david@khm.de; his website is alpha60.de.