SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 4, 171–179 (1974)

Implementation of a Management Game

R. E. THOMAS AND D. C. TOLL
Atlas Computer Laboratory, Chilton, Didcot, Berks., England

SUMMARY

The purpose of this paper is to describe the implementation of a management training game on two different computers. A brief description of the game is included, giving an idea of what the game looks like to the players and the controller. The two implementations are, firstly, on an Atlas I computer, with a multi-access system provided by an XDS Sigma 2 computer, and secondly on an ICL 1906A running under the control of the GEORGE 4 operating system. Mention is made of some of the difficulties encountered in each case, and of the techniques employed to solve these problems.

KEY WORDS Management game Interactive

INTRODUCTION

Management games are now widely used in the training of managers, and some of these games are controlled by computers. This paper describes the implementation of a game, originally designed to be controlled manually, on two different computers, together with plans for enhancing the second version when new facilities become available in the operating system. It highlights the different techniques required to do the same job under different conditions, comparing features offered by a remote job entry system and an interactive system which does not allow more than one terminal to be connected to a single program.

THE GAME

COLAB was designed by John Walsh[1] to provide a practical exercise for students taking part in one of the Science Research Council management training courses. There are, of course, many such games available, but most of these are geared to the running of industrial firms, whereas most of the course students here are more familiar with the research laboratory environment. In COLAB, a team of students find themselves in charge of a research laboratory which receives income from a number of sources, including the acquisition of contract work. The team has to pay for staff and equipment and allocate staff to contracts and to research.

Reasons for the choice of factors to be included are given in Reference 1. The game in no way tries to simulate the real world, since this would involve too many extra factors. There are sufficient recognizable details, however, for the students to have a 'feel' for what is likely to happen, and to practise the management skills that the rest of the course has taught. Further, it was decided that the game should be interactive, in that the game should be played with more than one laboratory, each competing for the same contracts, grants etc.; the actions of one laboratory affecting all the others. Thus the amount of grant received, for example, by a laboratory depends on various factors relative to the other laboratories.

Received 18 July 1973

Finally, definite formulae are used to provide numbers for such items as staff resignations. This removes most of the element of 'MONOPOLY' from the game, since random numbers are not used. A general description of these dependencies, although not including any detailed formulae, is to be found in the COLAB manual,[2] which includes a certain amount of 'local colour' material to set the scene, and the initial conditions for each laboratory all start with the same data.

COLAB works in 'quarterly' cycles, each covering a (simulated) three-month period. At the beginning of a quarter, the laboratory receives information as to how many staff have resigned, what grants have been made, what the current investment rate is for the laboratory's investment fund, and by how much its equipment has depreciated. It also receives a list of contracts on offer, together with the length of time each contract will last (in man-years). The laboratory now has to fix its salary and welfare levels (the second, notionally covering such items as pension funds, canteen etc., being expressed as a percentage of salary), and make bids for any of the contracts that it would like to undertake, by specifying an overhead in the form of a percentage of the total salary cost that the contract involves. The laboratory then receives a list of who has been awarded which contract (but not what each laboratory bid) and has to allocate all its staff to either the contracts it has won (either this quarter or in previous quarters) or research. Since the laboratory receives half its bid price on receipt of a contract and half on completion, and since there are penalty clauses for taking too long, it is in the laboratory's interest to finish contracts as soon as possible, but, against this, the level of research affects some of the grants received. The laboratory must also decide whether to recruit more staff (who will not become available for six months) or declare staff redundant (both operations costing money in training or redundancy pay) and whether to buy more equipment to allow for depreciation. Following these decisions, a balance sheet for the quarter can be prepared, showing how the laboratory is progressing.

CONTROLLING THE GAME

The previous section has given a general view of the game from the point of view of the student. However, there must also be a controller who awards contracts, calculates resignations, grants etc. and checks that student-prepared documents are correct. Originally, COLAB was designed without a computer in mind, so most of the tables, graphs etc. had to be compiled by hand. To the student, each quarter is split into separate actions roughly identifiable as bidding for contracts and allocating staff. However, having made bids, each team must wait while all other laboratories make their bids and also while control checks them, performs adjustment calculations on these bids according to relative salary levels, welfare levels and other factors, and awards the contracts according to the lowest adjusted bid. Similarly, having allocated staff to contracts, the team must wait for all other laboratories to make their allocations and for control to check them and calculate the new levels of resignations, grants etc. prior to issuing new contracts. Although waiting for other laboratories cannot be avoided, the amount of work to be done by control must be cut to a minimum.

THE FIRST IMPLEMENTATION

The Atlas Computer Laboratory was approached to implement COLAB. At the time, the laboratory had an Atlas I computer[3] with multi-access facilities provided by a front-end XDS Sigma 2 computer.[4,5] The satellite provided the control of the file store and the

terminals, but user jobs were submitted to Atlas for execution. The results of these jobs could either be output on the usual Atlas peripherals or returned to the file store for future interrogation. No user jobs ran in the satellite itself, and no interaction was possible with a job in Atlas. On such a system, COLAB was implemented.

The implementation can be divided into two parts: provision of programs to perform calculations and construction of a controlling system. The first deals with the calculation of grants, awarding of contracts, provision of balance sheets etc., and is machine independent in that these actions can be performed in the same way whatever control structure is used. This part was written in FORTRAN using free-format input and was readily transferable to another system (see below). However, the provision of a control structure involves making use of the facilities offered by the operating system on the computer, and is necessarily (with the current state of the art) very machine dependent. The rest of the paper will highlight the different control systems used in the various implementations employed so far.

The Atlas–Sigma 2 system, then, is non-interactive, and so some tricks have to be used to implement an inherently interactive system. It is also noted in passing that students who play COLAB have, in general, no experience of using computers, so unnecessary 'red tape' must be kept to a minimum (in this context 'red tape' means any typing or conventions that are not directly connected to the data being input or results being printed). The following features of the system were important in this respect.

(1) Command level macro facility.
(2) The ability to send messages from terminal to terminal.
(3) Timeout suppression (i.e. the removal of any system action caused by a user not typing within a given time limit).
(4) The ability to detect automatically when a job in Atlas finished.

The macro facility allowed many system commands to be obeyed by typing one pseudo command. This included the ability to type lines from a file (and hence cue input requests) and accept input from a terminal. Thus all the data for one run could be accumulated and the job to process the data submitted by typing one command only. However, since all data (such as, for one job, salary scales, welfare and contract bids) had to be available before the processing job was run, data errors were not picked up until later. This meant that separate macro commands for viewing the results of a job were needed in case a user had made a mistake and wished to correct it. Corrections were made by re-submitting all the relevant data which involved extra delays.

Messages could be sent to another terminal by using the command SIGNAL, which entered the message into a 'pigeon hole' file and also typed a short comment at the receiving terminal. The user at that terminal could then, at his convenience, use the MESSAGE command to read the message. In this way, each laboratory could inform control when some action such as bidding had been completed (i.e. data input, job run, results correct), and control could inform the laboratories when new contracts were available. This method of communication, of course, relies heavily on the fact that both the students and control must follow a strict sequence in use of commands. In practice this has worked quite well, since everybody co-operates fully on training courses (!), but mistakes have inevitably occurred.

One of the essential provisions of a multi-access facility which has more lines available than it can handle at once is the timing out of users who appear to have 'gone to sleep' (i.e. not typed anything for a predetermined period). However, COLAB necessarily generates long periods of thought, and any functioning of a timeout under these conditions is a nuisance. On Sigma 2, it was possible to suppress this feature for the duration of the game.

Since ATLAS sent a message to Sigma 2 when any job submitted by the satellite had finished, it was possible to tell each laboratory to check results as soon as they became available. Each job was very short, and so the time taken to control the game dropped considerably. Furthermore, the possibility of control making simple mathematical errors (very present under manual running) was eliminated.

Considering a typical three-month period, a player (i.e. a member of the laboratory team, as opposed to control) typed a request for contracts on offer. At the same time, he received details of resignations, grants etc. as before. He then typed a command which requested salary levels, welfare and bids, and ran a job to check the data. When the job finished, he typed a results file to see that all was well, and repeated the data if not. Finally, he sent a message to control.

Control waited for completion messages from all players before running a job to award contracts. On completion of this job, he typed the results and could, if he wished, run another job to make manual alterations to the awards (a rare occurrence). Messages were then sent to each player.

On receipt of the message, the player typed a request for award details. He then typed a command which requested staff allocation, recruitment, redundancies and equipment purchase, and ran a job to compile his balance sheet. When this job finished, and the data had been checked, he informed control by message.

Control, on receipt of all player messages, could now prepare for the next quarter by typing in the new contracts and running a job to calculate resignations, grants etc. for each player. When checking was complete, a message was sent to each player, who then repeated the cycle.

In this way, and, with the co-operation of all concerned, an 'interactive' system was provided, whose response time was quite acceptable. It demonstrates what can be done to run such a job on a system not designed to cater for that type of operation.

SECURITY

One general feature not mentioned so far is that of security. COLAB, as part of an organized course, must be run for the period stated in the course program (an evening and all next day), whatever the state of the machine. It is therefore essential to have safeguards against malfunction at every stage. As has been mentioned briefly above, a user could correct any mistake by re-running the job that performed the calculation. Since the jobs in Atlas appeared to be accessing normal FORTRAN I/O channels, it was possible to run the programs from cards, involving the punching of player data, if the terminal system went inoperable. Further, as the system was designed originally to run without a computer, it was always possible to revert to complete manual running at any stage if Atlas itself failed. Regular dumping of the system state on to backing store enabled temporary breaks by Atlas to cause minimal inconvenience, and, indeed, if the computer came back on the air after a prolonged absence, the current state of the game reached by manual means could quickly be set up for continued play at the terminal. In this way, security was assured.

SECOND IMPLEMENTATION

The second (and present) implementation of COLAB is on the Atlas Laboratory's new computer, a paged 1906A running under the control of the GEORGE 4 operating system.[6] This operating system includes full multi-access facilities, the terminals being handled via a message-buffering communications processor.

When the 1906A implementation was initiated, there were three stated aims. Firstly, it was decided to try to run the FORTRAN programs from Atlas on the 1906A, so as to minimize the amount of re-programming necessary. Secondly, the programs were to be made interactive, in that a player would type in data to a FORTRAN program when the program requested it, rather than input the data to a file and then connect the file to a background job (as was the case on Atlas). This enables the program to check for errors in the data as it is typed in, report on any as they are found and then offer the person responsible for the data a chance to correct it. Thirdly, in the Atlas system, in order to proceed with the game, the players (as well as control) had to type in commands to the systems. If a player typed in the wrong command, it was possible for various mishaps to occur, such as that player omitting part of the game. It was intended on the new system that either it would be made impossible for a player to type in an incorrect command for the current point in the game, or (preferably) the need for the players to type in commands at all would be removed.

Some problems were immediately apparent. In the Atlas implementation of COLAB, the controller and players communicated via the message facility of the Sigma multi-access system; GEORGE provides no similar facility. A second problem was possibly rather more serious; the COLAB game proceeds via a series of independent FORTRAN programs, some of which are run for control and some for the players. These FORTRAN programs access a common data area (held, on Atlas, in a disk file) so that a given program has a means of knowing the results of previous programs. In general, a program reads in this data, alters it and writes it back to the disk file. Further, when a FORTRAN program is to be run for a laboratory, one copy of the FORTRAN program is required per laboratory (since it was decided to use the original Atlas programs as far as possible) and these copies may only be run one at a time since all data is written to the same disk file. Hence if these FORTRAN programs are to be interactive under these conditions, one team typing in its data (and pausing to think about it) would cause the other teams to wait.

The chosen solution was to write a 'Range Compatible Trusted Program' (to use ICL terminology), which is in effect a form of operating system. This program is 'trusted' to run other programs (in this case the COLAB FORTRAN programs) under its control. The commands required by COLAB are then typed into this program, which can check that they are in the correct sequence. The controller's terminal was online to this control program, and thus the controller conversed directly with the system. There were no facilities in GEORGE 4 Mark 6 (the version current at the time of writing this program) to have more than one terminal online to a single program. To overcome this difficulty, each player's terminal had a small program online to it, and these small programs communicated with the control program via two basic peripheral communication files, one 'inward' and one 'outward'. Communication files appear to a program to be a card reader or a card punch and are organized such that one program can add to the end of the file while another program is reading data from the body of the file. The data put into the outward communication file by the control program had a control word on the front specifying which player should receive the line of data, and also containing a command to the program controlling the terminal such as 'print the rest of this line on the teletype' or 'read a line of data from the teletype and put it into the inward communication file'. Each player program could determine which lines from the control program were for its attention, and could also add the appropriate identification to the front of lines sent to the control program.

When a FORTRAN program is to be run for the players, the n copies (where $2 \leqslant n \leqslant 4$ is the number of players) are multi-programmed together. These programs are each run either to completion or to the point at which a program asks for data from the player; time sharing

could be implemented, since a trusted program receives timer interrupts, but it was decided that time sharing would involve unnecessary program swapping and would actually make the system slower. If a FORTRAN program terminates, it is marked as finished and the control program looks for another program that is free to run. If the FORTRAN program asks for input from the player, it is marked as awaiting input, and again the control program will attempt to run another.

When a FORTRAN program is being run, it must be positioned at the start of the 'Program Under Control' (PUC) area; further, when a trusted program is run under GEORGE 4, this PUC area has to be at least one quire (that is, 64 pages or 64K words) in size, and must start at a quire boundary (in this case, address 512K, since the control program is sparse—not all intervening pages are present). Hence when the current PUC is suspended it must be swapped out before another PUC can be run. Since the FORTRAN programs are approximately 9K words long, there are about 55K words of PUC area which have to exist but are wasted, and so this spare core area is used as a swap area. (Remember that we have to set up an operating system within an operating system!) It is also very much faster to swap into core than to swap programs out into a disk file. The swapping algorithm is such that swapping only occurs if necessary, so as to minimize the number of page turns.

The problem of simultaneous data access from the disk file is solved in the following way. The data is held in core by the trusted program, and when a FORTRAN program attempts to read from or write to this data area, the trusted program performs a core-to-core transfer. If the FORTRAN program is being run for a player, it will read data from the master copy held by the trusted program. However, when it writes the data block, the trusted program will generate a fresh data area for that player only. Thus when the FORTRAN program for each player has terminated, there will be the original (unaltered) master data area, plus one new data area per player, so it is possible to re-run the program for one or more players if necessary. After running programs for the players, the next stage of the game involves running a FORTRAN program for control, which reads in the data areas output by the previous player programs, and produces a new master data area to overwrite the original master data area. Thus, since this program's input data areas remain unaltered, it is also possible, if necessary, to re-run a program for control.

To make the COLAB system secure against a GEORGE crash or a COLAB system failure, or to enable the game to be stopped while the participants go to lunch etc., the master data area held in core is written to a disk file twice per cycle of the game. A new GEORGE file store file is used each time, so that previous files are not overwritten. Then, on a restart of the COLAB system, the latest file is read back into core, and the game continues from where it halted. Keeping the old disk files facilitates running a program to summarize the performance of each laboratory, since this program can obtain data at various points in the game from these files. Previously, all performance evaluation was carried out by the controller with paper and pencil during the evening following the game.

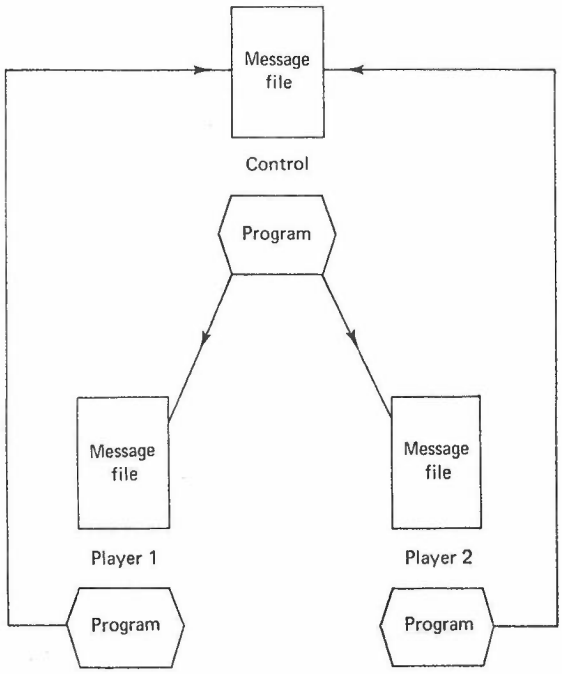
The system described above allows the FORTRAN programs to be interactive, by avoiding the disk file access problem. Further, the players only have to read what is printed on their consoles, and type in the data requested: otherwise, to the players, the game appears to run automatically. If the controller types in the commands in the wrong order, or types a command when the program is in the wrong state for that command, then the control program will flag an error and ask for another command. One disadvantage of the initial version of this system was that when FORTRAN programs were being run for the players, the controller could not interrogate the system to find out what (if anything) was happening.

Another was the fact that, if one of the teletypes becomes detached from the system for some reason (e.g. console 'inoperable'), the COLAB system crashed, and the game had to be restarted from a previous dump. A new version of the system, currently being tested, overcomes these difficulties. In this version, the controller's console, instead of being online to the control program, will be attached to a small program similar to the programs controlling the players' consoles. This additional program will communicate with the control program via the same communication files as are used by the players.

There are certain problems with this system. It involves the use of five programs at the same time (six in the latest version), which is a considerable drain on the 1906A's resources. Further, to ensure a rapid response at the terminals, the number of (non-COLAB) jobs in the machine has to be restricted. There are also some difficulties as far as the players are concerned, in particular, the fact that if they take too long to reply to a read request (that is, more than about 50 seconds), the console 'times out'. To get back to the previous state, it is necessary to 'break in', and hence it is necessary to explain extra features of GEORGE to the players. This timeout is particularly annoying here in that it is possible to effect a true 'break in' by pressing the key twice, which leads again to yet more detail that a player has to be told.

PROPOSED ENHANCEMENT

When GEORGE 4 Mark 7 is introduced, a new facility will be available to programs, namely the 'conceptual multiplexer'. This is a means whereby a program can online what appears to be a message buffering communications processor, but in fact is just a small part of the main computer's communications system. In this way a program can control a



Programs are started only after receipt of the appropriate messages

Figure 1. 1906A control system

multiplexor directly, while the usual GEORGE multi-access facilities are available on the remaining lines. It is intended that COLAB will use a conceptual multiplexor to control the five terminals. The latest version of the control program has been carefully arranged so that it uses communication files, but can be made to use a conceptual multiplexor merely by providing the extra code required in the input/output routine, the great majority of the program requiring no alteration. The coding for the communication files will be retained, since it is possible to input cards which look to the program like the lines of data in the communication file, and so the system can be run (albeit imperfectly) when the communications hardware is not working. Use of a conceptual multiplexor will alleviate some of the problems mentioned above, in that there will be only one program required to run COLAB. Further, any break-in can be dealt with by the program controlling the conceptual multiplexor, and hence it will be possible to avoid the added difficulties of restarting after a timeout.

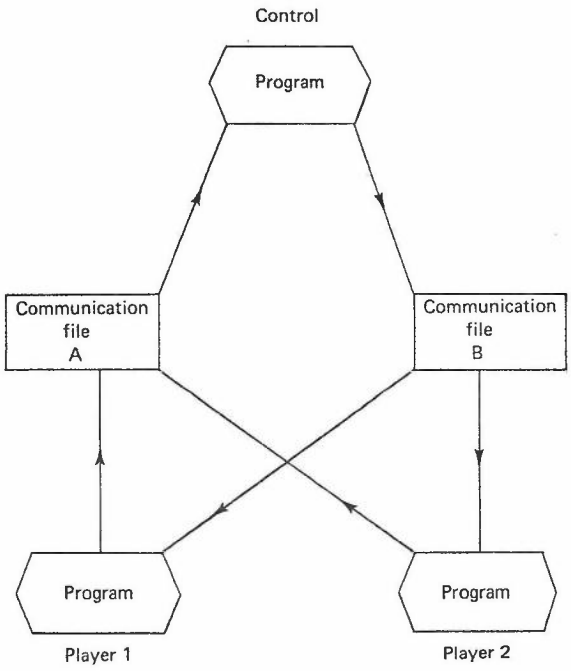


Figure 2. RŸE control system

CONCLUSION

This paper has shown how an interactive game can be implemented on considerably different systems, and what facilities in those systems help or hinder the work. It is interesting to note here that whereas the manual system managed to complete four quarters in the time available, the two computerized versions managed seven and ten quarters respectively. User reaction to the use of the terminal has been favourable, on the whole, although there is now a tendency to blame every player error and every delay on 'the computer'. In general,

however, the interaction with the terminal has helped to remove some of the inhibitions surrounding the use of such machines and has in fact managed to speed the game. At the interprogram communication level, most operating systems provide different facilities so that anyone wishing to run COLAB on a different machine will need to provide all such communication himself. However, the cases studied here represent a wide range of possible methods of attack, and should provide sufficient guidelines for similar exercises elsewhere.

REFERENCES

1. R. E. Thomas and J. D. Walsh, 'COLAB: designing a R and D management game', *European Training*, **1**, 133–144 (1972).
2. J. D. Walsh, *COLAB Manual*, Science Research Council Internal Publication, London, 1973.
3. T. Kilburn, D. J. Howarth, R. B. Payne and F. H. Sumner, 'The Manchester University operating system. Part 1', *Comput. J.* **4**, 222–225 (1961).
4. J. C. Baldwin and R. E. Thomas, 'Multi-access on the Chilton Atlas', *Comput. J.* **14**, 119–122 (1971).
5. J. C. Baldwin and R. E. Thomas, 'A critical evaluation of the Chilton multi-access system', *Software—Practice and Experience*, **2**, 313–320 (1972).
6. 'Operating systems GEORGE 3 and 4', *ICL Manual*, Technical Publications Service, ICL, Putney, 1972.