GAME: A LANGUAGE FOR WRITING BUSINESS GAMES

Leonard Fertuck

ABSTRACT

The GAME Language is designed to greatly simplify the process of designing business and other organizational simulations. A successful game requires not only a well-specified model with appropriate relationships between variables, but also a specification of input data, output reports, file structures and error checks. It is these latter components which are usually the most tedious to code, the most poorly coded and the most standard in requirements. The GAME Language is designed to simplify or eliminate these functions while maintaining a great deal of flexibility in model specification.

A generator converts free-format specifications in the GAME Language into a PL/I program. Options permit Monte Carlo and Sensitivity Analysis and corporate simulations.

INTRODUCTION

Business games have been used for more than a decade to provide a simulated environment in which students can practice decision-making. There are literally hundreds of them in existence, each one having been designed to model a particular environment and illustrate a particular set of decision problems. While all are different in various respects, most share many similarities.

The problems common to all computer operated business games are the need for editing input, updating historic parameters, maintaining historic data files, and administering input decisions. These components typically account for well over half of the code in most business game programs. Yet these functions are very poorly organized in most business games.

A business game is typically designed and programmed by someone with no previous experience in designing business games. Their interest is usually in developing a model to illustrate some concept. The data-handling components are treated as a necessary evil and little thought is given to the problems of checking errors and maintaining multiple files for multiple classes. As a result, a large workload that could be automated is left to the game administrator to handle manually. These problems are difficult to eliminate in any given game because games are generally complex and difficult to modify. The addition of a single decision parameter in the game can effect the input section, the output reports, the reading, writing, and updating of history files, the model computations, and the data definitions in each routine. As a result, the game designer tends to avoid modifications and improvements because they will require a great deal of work and will be very error-prone. A few attempts at modification will leave him so frustrated that he will not only avoid further modifications, but will avoid having anything to do with designing business games in the future. As a result most designers design only one game and most games are designed by inexperienced designers.

These problems can be greatly reduced or eliminated by providing the designer with a computer applications language which could provide an organized framework for designing games and relieve the designer of the necessity of reproducing designs for standard operations such as file handling and report formatting. Such a language would produce a program in which all file input, file output, and history updates are automatic and require no specifications from the designer. Standard features would be provided for editing input cards and formatting output reports. Many standard checks would be performed for common input errors. Since these operations would standardize many of the administrator's functions, a large part of the administrator's operating manual could also be pre-written.

The designer would be left with the task of specifying the variables in the game, the relationships between these variables, the data which are to be provided by the players, the data which are to be provided to the players, and the playing rules. This is still a significant task, but it is one which the designer can perform. Testing of alternate designs is made much easier by the use of an applications language, since many of the time-consuming operations in a modification are performed automatically. Since testing and redesign would be much easier with such and aid, it is likely that more improvements would be incorporated before a design is finalized. Since the process would be much less frustrating for the designer, he would be more likely to attempt other designs which would benefit from his previous experience.

THE GAME LANGUAGE

The GAME (Greatly Augmented Modeling Environment) Language eliminates the need for coding of standard functions and allows the designer to concentrate on his design. The language processor translates the User's Model into a PL/I program which can then be compiled and executed. The translator is also coded in PL/I. The GAME Language assumes that a model will have one or more industries whose data are stored on separate files, and whose decisions do not interact in any way. This is a convenience in administering games since input for several independent groups of players, possibly at different stages of plays may be batched together. Each industry consists of one or more firms whose decisions may interact. All parameters for the industry are simultaneously available. Decisions or plays in the game are made periodically. If decisions are not submitted or some decisions are left blank, previous decisions are automatically repeated on the assumption that a continuing policy has been established. This reduces input preparation for the players and makes administration easier. The number of firms in an industry can be dynamically set during the initial period of play.

Each firm may be designed with multiple subcomponents which may represent regions, factories, products, or other factors. The number of these can be easily changed but must be the same for all firms.

There is no inherent limit to the number of periods which may be played. Data for 1 previous period is always available for use in lagged relationships or as a backup in the event of processing failures. Multiple periods may be processed in a single run.

The GAME Language has 8 distinct blocks: OPTIONS, LIMITS, GLOBALS, FIRMS, TEMPS, INPUT, COMPUTE, and OUTPUT. Each block must start with the keywords DEFINE block and end with the keywords END block where block is replaced by one of the block names. Blocks can be omitted if not needed, but they must be in the specified order if they are used.

In describing the syntax of the blocks, the following notation will be used:

a) Keywords are shown in capitals and user-defined variables are shown in lower case.
b) Square brackets ([]) denote optional elements of the language.
c) Three dots (...) denote that the preceeding element may be repeated one or more times.
d) A vertical stroke (|) indicates that one of the elements separated by vertical strokes must be chosen.
e) , : ; ( ) and blank are used as delimeters and extra blanks may be inserted for readability.

BLOCK DESCRIPTIONS

1. OPTIONS

The OPTIONS section is used to specify which of several optional output features and program features will be used during this run of the GAME language. The syntax is: DEFINE OPTIONS [ATTRIBUTES] [INPUT] [SPLIT (m)] [VARIABLES (n)] END OPTIONS

Only the first letter of each option name is significant. Thus ATTRIBUTES, ATTR, A or AARDVARK have the same effect. The default options are that none of the outputs are provided.

ATTRIBUTES will cause, a table in alphabetic order to be printed at the end of the run listing all of the variable-names and their initial values, dimensions, extended labels, and upper and lower legal values.

INPUT will produce a table of the columns in which the input data variables are to be punched on the decision cards submitted by simulation players.

SPLIT (m) allows the translation to distinguish between input data provided by the player and input data provided by the administrator. A card number equal to or greater than m is treated as administrator reports to warn that a change of parameters has been made. The default value of m is 100 implying that all input is treated as player input. See the INPUT section for more details on the use of this parameter.

VARIABLES (n) allows the programmer to change the size of the internal tables used to store variable names and their properties. The default value of n is 100. Any increase in the size of n will increase the REGION needed by the computer to operate the translator. About 300 bytes of storage are needed for each variable.

2. LIMITS

The LIMITS section is used to specify the dimensions and labels of any repetitive factors in variables to be defined later. The syntax is:

```
DEFINE LIMITS
[name: label [,label]...;]
END LIMITS
```

Up to 9 names may be defined and each may have as many labels as required. Each label will increase the maximum subscript number of the named factor by 1. As an example, REGION:EAST, WEST will make provision for 2 regions in any variable whose SIZE in the next section is defined by REGION.

3. GLOBALS, FIRMS, and TEMPS

The GLOBALS, FIRMS, and TEMPS sections are used to define the properties of all variables used in the program. The three sections are identical

except for the way in which the specified variables are used and stored. Variables specified in the TEMPS section are not stored in the history files and may not be changed by input cards. They are intended only for use as intermediate variables in calculations, or as variables computed for use in reports. Variables specified in the FIRMS and GLOBAL sections are saved for reuse in future periods. Thus a lagged value is available for each of them. FIRMS variables are automatically given a dimension equal to the number of firms in the industry. GLOBALS variables are not given any automatic dimension and are used for factors which apply identically to each firm.

The syntax is:
DEFINE section
[structure-level variable-name [option]... ;]...
END section

where:
section is FIRMS|GLOBALS|TEMPS
structure-level is a positive integer between 4 and 10
variable-name is a name not previously defined
option is any of:
HI(value)
LO(value)
INIT(value [,value]...)
LABEL(character-string)
SIZE(name [,name]...)

The rules for structure levels are the same as those for PL/I except that the first 3 levels are internally defined by GAME.

The SIZE option converts the element or structure to which it is applied into an array of elements or structures with dimensions defined by name in the LIMITS block.

INIT (value[,value]...) allows the designer to specify initial values for a variable. The default value is zero for all elements. If there are not enough values to fill an array, they will be repeated from the beginning until enough values are obtained. Arrays will be filled by cycling the last subscript fastest.

HI (value) can be used to specify a maximum allowable value for input variables. The default value is 1E75. If an input value exceeds the HI value, it is reset to the HI value and a warning message is printed.

LO (value) can be used to specify a minimum allowable value for input variables. The default value is -1E75. If an input value is less than the LO value it is reset to the LO value and a warning message is printed.

LABEL (character-string) can be used to give an extended label to variable names. This label will be used on all output reports on which this variable is printed. The character string must not exceed 30 characters. Leading blanks and excess characters are deleted.

4. INPUT

The INPUT section defines the format of input data cards which are used to submit decisions and parameters to the simulation. This section is not required in simulations, such as growth models, which operate entirely on initialized values. The syntax is:
DEFINE INPUT
[CARD(n): [@][variable(column,digits)]...;]...
END INPUT

n must be between 1 and 97. Cards do not have to be defined in numeric order. The column number must be greater than 8. The @ symbol enables the input of array cross-sections as described later.

All data cards have the following fields predefined, and each field must be punched on each card.

col 1-2 = Industry number. Industry numbers may be between 1 and 99 and must be punched on every input card. A check is performed for illegal numbers above a user defined value so the lowest possible numbers should be used.

col 3-4 = period number. Input for period -1 initializes the history files for an industry. Period 0 can be used to provide the first printed report to be distributed to players so that they can see the initial status of their firms. Subsequent periods are then played in numeric order.

col 5-6 = firm number. Firms are numbered from 1 to N in each industry. N is defined for the industry during period -1 and may not be changed. The definition is performed using an authorization card to be described later in this section. Administrator input, which applies to no individual firm, should have a blank or zero in this field.

col 7-8 = card number. This field identifies the format by which the card is to be read. This is the number referred to in the CARD(n) portion of the DEFINE INPUT section. Card numbers must be positive 2 digit integers. The format of cards 0, 98, and 99 have been predefined.

The above fields uniquely identify every input data card and are arranged so that sorting the cards by these fields will keep all cards for a single firm, period, and industry together. Since the simulation automatically performs this sort, input cards may be submitted in any order by the game administrator.

Three cards with special uses have been predefined. Card 0 is an authorization card used by the administrator to authorize play for each period. Card 98 is used to broadcast NEWS messages to all teams in an industry. Card 99 is used to enter industry and firm names and control the number of copies of each report that are printed.

Card 0, the authorization card, is needed to allow the game administrator to set certain control variables each period and also to insure that

players do not accidentally initiate play for one or more periods by a keypunching error. All cards for an industry and period will be ignored if an authorization card has not been submitted for that industry and period. The firm number must be left blank or zero on this card. The format of the card is:

col 11-19 = Flags. These flags are either zero (or blank) or one. These 9 flags can be used to suppress reports specified in the OUTPUT section. Reports will be printed if the corresponding flag is set to zero and suppressed if the flag is set to one. This is a simple way of inhibiting needed reports or controlling reports that are required intermittently such as every 4 periods.

col 20-21 = number of firms. This field is needed only during period -1 when the number of firms must be specified so that arrays and files of appropriate size can be created. It will have no effect in other periods.

col 22-23 = firm number to be saved. The current history of 1 firm and of the GLOBALS variables can be saved on file SAVE to be used to subsequently initiallize all firms in an industry. The initiallization may occur in an industry with a higher number during the same run or in any industry during a subsequent run if the SAVE file is permanent. This parameter will have no effect if it is zero or an illegal firm number. Subsequent saves will overwrite the file which has been created. This feature is very convenient for storing standard initial values or using an old result as a new starting point.

col 24 = code to get file SAVE. If this field is nonzero the data saved by a previous use of col. 22-23 will be copied into all FIRMS and GLOBALS variables. It has an effect only during initiallization in period -1. The copying takes place before a save on the same card is performed.

Card 98 is used to generate a file of messages to be printed for the administrator and each firm. Any character string may be entered in columns 11-80. Up to 99 lines of message may be submitted by the administrator and each of the teams. Each line must be numbered in the index field in col. 9-10 of the data card beginning with the index 1. All 5 fields in cols. 1-10 must be punched. The message will be printed only in the specified industry and period. This is a convenient way of distributing administrative information or advertising firm information such as willingness to sell surplus product.

Card 99 is used to enter an industry name if the firm number is zero or a firm name if the proper firm number is specified. The name is entered without leading blanks in cols. 11-78 and is centered and printed on all reports. If this card is not submitted, the name fields on the reports remain blank.

The designer should arrange all of the input data on the card so that data to be provided by the players is on the first cards and data to be provided by the administrator follows. The SPLIT option in the OPTIONS section will generate code to check which kind of data is being read and issue a warning when administrator data is being read. This will warn the administrator if a player accidentally submits data which changes parameters he is not entitled to change.

If input fields are filled with blanks, the simulation automatically uses the value from the previous period as stored in the history file.

All input is checked to ensure that it is within the allowable range specified in the HI and LO options of the variable definitions. If it is not, a warning message is issued to the administrator and the variable is set to the nearest allowable value. Input data may be entered as integers, decimals, or scientific numbers.

Under some circumstances, cross-sections of arrays may be filled with one entry. This is particularly useful if the administrator wishes to set all firms, products, regions, etc. to have the same value. This feature can be enabled only for the administrator data above the SPLIT card.

The designer enables the cross-section feature by preceding the desired variable with an @ symbol. When this is done, the regular input routine is replaced by a routine which cycles through all defined values of any firm number or index number which has been set to blank or zero. Thus if X is an array of 3 variables for N firms, a data card with index number=2 and firm=0 containing a value of 10 for X will have variable 2 set to 10 for each firm. The sorting of input cards causes cross-sections to be encountered before individual values. It is therefore possible in the above example to submit another card to give a value of 20 to team 2 only. When cards with identical identification fields are submitted, the last one read overrides previous ones, but the sorting order cannot be predicted, so the result is not predictable.

The column parameter on the input specification indicates the card column in which data for the specified variable begins and the digits parameter indicates how many columns are reserved for this variable. The column number on subsequent variables must be greater than the column numbers used on any previous fields.

5. COMPUTE

The COMPUTE section is used to specify computed relationships between variables. Array and structure computations are possible. Facilities are provided for including PL/I code to perform loops and logical tests. The syntax is:

```
DEFINE COMPUTE
[[FOREACH|FORALL]:variable =
value[relation value]...;["PL/I code"]]...
END COMPUTE
```

where
a) variable is a variable or structure name defined in the FIRMS, GLOBALS, or TEMPS section.

b) value is a variable, a constant, or a function reference of the form function (value). All variables in a syntactic line must have the same dimensions and structure. A variety of mathematical functions are provided.

c) relation is one of the following:
+ for addition
- for subtraction
/ for division
* for multiplication
** for exponentiation

d) PL/I code is any syntactically correct PL/I code. Since GAME performs no syntax checking, errors will not be detected until the PL/I program is compiled.

FOREACH specifies that a loop is to be constructed to compute values for one team at a time until another FOREACH or FORALL closes the loop. This will produce some saving in execution code and will allow the insertion of PL/I code to perform individual handling of firms.

FORALL specifies that the computations will be array or structure operations applying identically to all firms. Effectively, a loop is generated around each syntactic line.

Lagged values of variables obtained for the previous period from the history file may be obtained by preceding the variable name with the structure name LAG. Thus, a change in X can be computed as shown below:

```
CHANGE=X-LAG.X;
```

optionally, the current period may be preceded by the structure name NOW., but the translator will insert this if required. Only FIRMS and GLOBALS variables may be lagged and only a single period may be lagged. Situations where multiple period lags are required can usually be handled by an exponential lag equation of the form:

```
X=a*LAG.X+(1-a)*X;
```

where a is between 0 and 1. The effective lag period increases as a increases.

PL/I code can be included after a ; in the COMPUTE section by enclosing it in double quotes ("). This is not the same as using two consecutive apostrophes. Anything between double quotes is inserted without syntax checking, so it is important that the code be correct.

6. OUTPUT

The OUTPUT section is used to specify the variables which will be printed on reports to the administrator and to the players. Automatic formats and headings are provided. Four kinds of reports are available. The ECHO report will list the current value of all input variables on a report for each firm so that input can be verified. The LIST report will list the variable label, any subscripts, and the value of the element in single columns in the order specified. The TABLE report will perform the same function in multiple columns, one for each team. It is most useful for listing comparison figures for the administrator or releasing public data, such as financial statements, to all teams. The NEWS report is used to print messages submitted on CARD (98) of the input. The syntax is:

```
DEFINE OUTPUT
[[option]...:[variable]...;]...
END OUTPUT
```

where option is one of the following (defaults are underlined):

```
HEADING(YES|NO)
FLAG(0|1|2|...|9)
FORM(LIST|TABLE|ECHO|NEWS)
FILE(PLAYER|ADMIN)
TITLE(blank|character-string)
```

and variable is an element name or structure name or format control. A format control is one of LINE, SKIP, or PAGE. The options can be used to control format and direct reports to desired files.

TITLE (character-string) will define a title of up to 70 characters to be inserted in the standard heading. If the option is not used, this field will be left blank.

HEADING (YES|NO) determines whether a complete heading is printed at the top of the report, or only the title from the TITLE option. If a complete heading is printed, it will start at the top of a new page and include, the industry, firm, period, and page numbers, and the date, industry name, firm name, and title. If the heading is not printed, a title is printed two lines after the end of the last report on the same page. If the end of a 60 line page is encountered, a heading will be printed automatically at the top of the next page regardless of which option was selected.

FLAG(n) determines which of the 9 flags on the authorization card will control printing of this report. If the corresponding flag on the authorization card is set to 1, instead of 0 or blank, the report will be suppressed. If this option is not used, the default is to use a hidden FLAG(0) which is set to always print the report.

FILE (PLAYER|ADMIN) determines whether the report will be printed on the PLAYER or the ADMIN file. This option is provided so that PLAYER reports can be printed on multipart paper without also printing ADMIN files on multipart paper. The default is to print on the PLAYER file.

FORM (LIST|TABLE|ECHO|NEWS) determines the format and content of the output report. The ECHO report produces a report on the current value of all data items on cards with numbers below the SPLIT level in the OPTIONS section. A report is created on the PLAYER file for each firm and contains only data for that firm. No variable list is required since all variable names are obtained from the INPUT specifica-

tions.

The NEWS report is a listing of any messages submitted by the administrator or players on CARD (98) of the input. If FILE(ADMIN) is specified, this becomes a means for players to send messages or complaints to the administrator. This report does not require a variable list.

The LIST report produces a 1 column listing of labels (from the LABEL option of the variable definitions) subscripts and element values under a standard heading and title. The variables to be listed are taken from the list following the colon. If an array is included in the list, it is printed with 1 element per line, cycling the last subscript fastest. If a structure is listed, each element of the structure is listed in order.

The TABLE report produces a table with variables listed vertically and firms ordered horizontally. Thus, there is a column of data for each firm. If there are more firms than columns on a page, the report will be continued on the next page.

LINE will cause an underline to be printed under the most recent value. SKIP will cause 1 blank line to be inserted in the output. PAGE will skip to a new page.

IMPLEMENTATION

The GAME Language is currently being implemented in PL/I. While this choice of language limits its use to IBM installations, it also makes development a great deal easier because many of the powerful features of PL/I such as data structures can be incorporated directly into the language without a great deal of complex coding.

The translator consists of a separate subroutine to handle each one of the language blocks. The GLOBALS, FIRMS, and TEMPS blocks are used to construct a data structure which defines all user variables and all internal variables. Internal variables always start with the @ symbol to distinguish them from user defined variables. Tables of HI and LO values and LABELS are also constructed for later use. The data structure is then copied into the main program and later into subroutines which perform INPUT, COMPUTE, and OUTPUT functions when called by the main program. These subroutines are automatically overlaid to conserve core storage.

The main program performs all file operations by reading and writing substructures. It is also capable of bypassing computations and reports so that input can be economically checked for errors.

The INPUT block is translated into an INPUT subroutine which reads all data for a single industry and period. An internal routine checks each date field for legal non-blank values and inserts them into the database. When input has been completed, the updated database is passed on for computation.

The COMPUTE block is the weakest part of the language since it only allows limited array calculations and no conditional statements. An IF-THEN-ELSE capability is being contemplated for the future.

The OUTPUT block is translated into a series of internal subroutines with one routine for each report. These in turn are called sequentially by another routine which first performs standard operations like printing headings and titles.

The translator also has routines to list some of its internal tables. One lists the variables and their dimensions and labels which is useful for debugging. Another lists input variables by card and column. This list is very useful in producing player manuals.

One of the advantages of using the GAME Language is that most of the procedures followed by both the players and the administrator are very standardized. This means that a large part of the documentation which is required in the Player's Manual and the Administrator's Manual can be taken directly from the GAME Language Manual. This eliminates a large part of one of the most onerous chores of model implementation.

FUTURE EXTENSIONS

Since a corporate simulation can be thought of as a business game with only one team, the GAME Language could be used for corporate simulations if it were provided with some of the commonly used functions such as depreciation and discounting. It would also be relatively easy to add some random number generators, add some different reports and modify the main program to permit the iterations required for sensitivity analysis and Monte Carlo simulation. These features would also be very useful to the game designer because it would make it very easy for him to test the stability of his model under a variety of inputs.

SUMMARY

The GAME Language provides a simple tool for the design and implementation of business or other interactive organizational games. The tedious chores of documentation, variable definition, file handling, error checking, and input or output formatting have been greatly reduced or eliminated. This allows the designer to concentrate on designing a good model and makes it easy to improve the model.