Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

SIMSCRIPT II.5 and SIMGRAPHICS: A tutorial

Edward C. Russell
CACI Products Co.
12011 San Vicente Blvd.
Los Angeles, CA 90049

ABSTRACT

The SIMSCRIPT II.5 programming language is described, and a complete simulation example is presented.

INTRODUCTION

SIMSCRIPT II.5 is a well established, standardized, and widely used language with proven software support. Experience has shown that SIMSCRIPT II.5 reduces simulation programming time and cost several fold when compared to FORTRAN. It assists the analyst greatly in the formulation and design of simulation models and gives the programmer and analyst a common language for describing the model. The benefits of using SIMSCRIPT II.5 can be felt at all stages in the development of a model, including:

Design: The powerful "world-view" consisting of Entities, Attributes, and Sets provides a natural conceptual framework in which to relate real objects to the model.

Programming: The modern, free-form language contains structured programming constructs and all the built-in facilities needed for model development. Model components can be programmed so as to clearly reflect the organization and logic of the modelled system.

Testing: A well-designed package of program testing facilities is provided. Tools are available to detect errors in a complex computer program without resorting to memory dumps and other archaic means.

Evolution: The SIMSCRIPT program structure allows the model to evolve easily and naturally from simple to detailed formulation as more information becomes available. Many modifications, such as choices of set disciplines and performance measurements are simply specified in the program preamble in a non-procedural manner. Animation and presentation graphics can even be changed without program modification.

Documentation: The powerful English-like language allows for modular implementation. Because each model component is readable and self-contained, the model listing can be understood by the end-user who may not be at all familiar with programming. Because the detailed model documentation is the program listing, it is never obsolete nor inaccurate.

OVERVIEW

Purpose of Simulation

The purpose of a simulation must be clearly articulated before embarking on model development. Many modelling efforts have been doomed to failure, because a clear goal was never determined. The natural tendency is to model in great detail that part of the system which is well understood and to "sweep under the rug" or over-simplify those parts which are not understood. The detailed model of the well understood parts yields many lines of model code and gives the illusion of great progress, when in fact, a much smaller model of the entire system may actually be of much greater value. In general, a model is an abstraction of the real system under study. It is not necessary or even desirable to include all of the details of the actual system. Deciding which details are essential and which may be omitted for the purposes of the study is perhaps the most difficult task which the modeller faces.

Concept of a World-View

Without its "world-view", SIMSCRIPT II.5 would be just another programming language, albeit a very powerful one. But with its world-view, the modeller is guided in the formulation of a complete specification of the problem. The objects in the real world map very naturally into the SIMSCRIPT II.5 objects which break down into classes termed TEMPORARY ENTITIES, PERMANENT ENTITIES, PROCESSES, and RESOURCES. (All capitalized words are part of the SIMSCRIPT II.5 vocabulary.) Any entity may have ATTRIBUTES which give it individual characteristic values. While all instances of a particular entity class have the same named attributes, each instance has its own values for the attributes. In addition, entities may be organized into SETS in order to represent any type of ordered list with various ordering disciplines (FIFO, LIFO, or RANKED by any combination of attribute values).

After the static structure of the model has been described, the dynamic aspects are described in terms of process routines. Each process routine corresponds to a declared process entity. Very natural commands are employed for manipulating objects in the process routines. Processes may WORK or WAIT for a period of simulated time. They may be FILEd in sets or REMOVEd from them. They may ACTIVATE, INTERRUPT, or RESUME one another. Processes may REQUEST or RELINQUISH resources, automatically waiting for those which are unavailable when requested and automatically other processes when relinquishing unneeded resources.

Animation in SIMSCRIPT II.5 is a very natural extension of the established world-view. Entities may be declared to be GRAPHIC in order to participate in graphic displays. They may be further expanded to be DYNAMIC GRAPHIC entities in order to participate in animated displays. The actual form of the display (the so-called "icon") is described through the use of an editor and may be changed independently of the model.

Self-Documenting Code

Over the years, we have observed numerous unsuccessful simulation projects that had no documentation except a FORTRAN listing. Many of these listings contained few explanatory comments. Even a thoroughly commented FORTRAN listing is difficult to decipher for anyone other than the person who wrote it. Often, even the original author has difficulty understanding it after a short time.

We have also seen great amounts of money wasted on manuals and flowcharts intended to make it easier to develop, maintain, modify, and enhance the model. This waste is a consequence of the realities of model development. Most models evolve over a long period of time because of new and increased understanding of the system, changing goals and availability of new data. Because of the evolutionary changes, flowcharts, prose documentation, detailed descriptions of routines and variables, and program comments often become obsolete, incomplete, or incorrect shortly after they are written. The longer the model is around--and many models in use today were developed five or more years ago--the more this type of documentation deteriorates.

For the purposes of computer program development, modification, and enhancements, the only dependable documentation in a changing environment is the source program listing. The quality and usefulness of this documentation is determined by the model design and the choice of simulation language. SIMSCRIPT II.5 has been shown to reduce the amount of code required when compared to FORTRAN by at least 75 %, a four to one reduction!

Large model development

SIMSCRIPT II.5 has traditionally been the language of choice for very large models. There are no inherent limits to the size of either SIMSCRIPT II.5 programs or their data structures. The dynamic storage allocation of SIMSCRIPT II.5 frees the modeller from concerns about the size of data elements and all the error-checking code necessary to enforce array limits. The modularity of the language structure permits large teams of developers to work on independent segments of the model without needing to know all of the details of other elements of the model.

Portability

SIMSCRIPT II.5 is a truly portable language. It was originally developed for large mainframe computers, but it has evolved with the industry to implementations on mini- and now micro-computers. The IBM Personal Computer implementation of SIMSCRIPT II.5 is one of the most advanced software packages available on that computer. The modelling language has been designed and maintained to be compatible across all the implementations. These include: CDC, Cray, Datageneral, ETA, Gould, Honeywell, IBM, Prime, Sun, UNISYS, VAX (both VMS and UNIX).

SIMSCRIPT II.5 LANGUAGE FEATURES

SIMSCRIPT II.5 is a complete programming language. In addition to its simulation modelling capabilities, it has a full range of input / output capabilities including the ability to specify either formatted or free-form input, screen-oriented output (including cursor placement), generalized reports which may expand to multi-page width as well as length. The TEXT mode of variable declaration permits very general text manipulation of character strings of arbitrary length including operations such as concatenation, substring search and replace, case change, etc.

The entity / attribute / set structure mentioned above is an extension of a very powerful underlying data structure. Arrays in SIMSCRIPT II.5 may be of any dimension whatever, without limit. The allocation of storage for the arrays occurs during execution and arrays may be deallocated and reallocated with different dimensions. (If there were a need for much reallocation, the temporary entity concept would more likely be used.)

SIMSCRIPT II.5 contains all of the constructs of modern structured programming. Search commands relate to the data structures to be scanned. Program segments may be modularized along functional lines as routines, functions, monitoring routines (to be called implicitly when the monitored variable is either accessed or modified or both), as well as the process and event routines of simulation. Routines may also be executed recursively.

The support of the representation statistical phenomena is extensive. Generators exist for random numbers distributed according to uniform, integer uniform, normal, log normal, exponential, beta, gamma, erlang, poisson, binomial, triangular, and weibull distributions. If these are sufficient, an arbitrary numerical distribution is available to describe any distribution as a table of values versus probability (individual or cumulative).

The collection of data in the form of statistical performance measures is supported by three very powerful statements; ACCUMULATE, TALLY, and COMPUTE. ACCUMULATE and TALLY are declarative statements which prescribe what measures to take but not how to accomplish them. COMPUTE is an executable statement which performs the computations at the time it is executed. Whereas ACCUMULATE and TALLY update statistical counters as the variable of observation changes values. Then only when the results are needed are the final statistical calculations performed. The measures available include number of samples, sum, average, maximum, minimum, standard deviation, variance, sum of squares and mean square. ACCUMULATE performs these calculations on a time-dependent basis, while TALLY performs them on a sample-basis.

SIMSCRIPT II.5 has recently been enhanced to enable the user to develop models which include processes which change continuously with simulation time. This enables models to be built for those systems which are described in terms of differential equations with superimposed discrete events. The combined capabilities enable the user to define models where dependent variables may change discretely, continuously, or continuously with discrete jumps superimposed.

Part of the ongoing development effort of SIMSCRIPT II.5 is to make the interface between user and model easier to understand. Traditionally, the output of a simulation run was collated tables of data which required extensive analytical capability on the part of the user in order to understand the underlying interactions between various parts of the system under investigation. Much progress has been made in providing facilities within the language whereby these interactions may be represented graphically. This enables models to be developed in which the parameters can be easily represented as presentation graphics such as pie charts, strip charts, dials, level meters, bar graphs, etc. These so-called "smart icons" are updated on the screen as the simulation proceeds. In addition, animation capabilities have been developed to display moving objects against a static background in order to give further insight into the complex interactions which take place within a system.

The preparation of the presentation graphics as well as the icons for animation is accomplished through the use of editors. The icons are stored with the program but may be modified without need to modify the program or clutter it with non-system-related code. At present, these facilities are available only in the PC version of SIMSCRIPT II.5, but development is underway for several other implementation.

THE PC SIMLAB ENVIRONMENT

Many of the capabilities of PC SIMSCRIPT II.5 are made possible because of SIMLAB. SIMLAB is an operating environment for the SIMSCRIPT II.5 compiler, editors, and run-time. SIMLAB supports a multitasking environment in which it is possible to perform several tasks simultaneously. During program development, it is possible to edit several portions of the program simultaneously (in different windows). During execution, it is possible to open "debug windows", set break points, and track the execution of the program through its source code. It is also possible to have multiple, concurrent output streams to different windows. SIMLAB also makes it possible to write, maintain, and execute programs which are much larger than the actual memory of a PC can contain. Through its virtual addressing mechanism, programs which would normally require main frame capacity are being developed on PCs.

A TUTORIAL EXAMPLE

As an example of model building in SIMSCRIPT II.5, we shall now construct a model and discuss its components as we proceed.
The problem is one which has appeared from time to time in the literature (origin unknown).

> Consider a mining operation which uses a lift to carry ore from the underground levels to the surface. There are two underground levels where level one is the deepest. Level one is the most productive level, so the lift always descends directly to it if ore is available there. The decision to go deeper is made once the lift approaches level two on a descent. If neither level has ore ready for transit then the lift descends to level one and waits for ore to be produced there. Assume the following characteristics for the lift system:
>
> 1. The lift has a capacity of three loads of ore.
> 2. Each level produces an average of five loads per hour distributed according to a Poisson process.
> 3. The lift requires three minutes to travel between levels except that an extra thirty seconds is required to stop at level two. This thirty seconds is applied both to the movement to and from level two.
> 4. The time to place one load of ore on the lift is uniform over the interval (0.75,1.25) minutes. The time to remove one load of ore from the lift follows the same distribution.
>
> The system starts empty and idle with the lift at the surface. Estimate the average content of the lift as it approaches the surface. Also estimate the average and maximum queue length at each level. Run length will be an input parameter.

The SIMSCRIPT Approach

The first step in the model design is to specify the components of the model. In SIMSCRIPT II.5 these very closely relate to the objects of the real system. The PREAMBLE (figure 1) contains the static description of the model. The lift is the dynamic player in the system and will be modelled as a PROCESS. The loads are passive. They appear randomly on each level, queue for the lift, and eventually are loaded and carried to the surface. We model them as TEMPORARY ENTITIES. In this way we do not need to be concerned with the number of loads in the system at any given time. SIMSCRIPT II.5 will allocate and deallocate storage for them as needed. Certain information is to be stored about each level in the mine. Since the number of levels is fixed, we model them as a PERMANENT ENTITY with two copies. Two somewhat artificial processes are defined to start and stop the simulation. The first is LOAD.GENERATOR which will generate loads at the required rate for each floor. The second extra process is FINAL.REPORT which will print the final results and stop the simulation at the specified time. The attributes of each of these various entities should be self-explanatory. A SET is used to contain the loads waiting for the lift on each floor. This set is described as OWNED by each floor and BELONGED TO by each load. The set discipline is declared as LIFO although it should really be FIFO. The reason for this was to speed up the animation of the model by only deleting and redrawing loads at one end of the queue rather than moving them up one space at a time.

The remainder of the PREAMBLE describes performance measures and graphic controls, each of which will be described below.

The Main Program

The main program (figure 2) is where execution begins in a SIMSCRIPT II.5 program. In our example, we have modularized the program along functional lines such that the main program consists of a series of subroutine invocations followed by the start of simulation.

Each routine will be described in turn. After the initialization has been completed, START SIMULATION passes control to the timing mechanism. This is the heart of discrete simulation. The timing routine sequences the execution of the processes according to the "next event" policy. That is, a list of currently scheduled processes is automatically maintained, ranked according to the scheduled time of execution (in simulation time units). As each process comes to the head of the list, it is removed and executed. If the process was waiting for a passage of time (e.g, in a WORK statement, execution resumes at the next statement after the WORK statement. The timing routine continues to sequence processes until either the list is empty or one of the processes stops the simulation. (In our case, FINAL.REPORT will stop the simulation.)

Initialization Routines

The bulk of the code in this model is concerned with setting up the initial conditions. First a set of default values are established in routine SET.DEFAULTS (figure 3). Then the user is offered the opportunity to change many of these values in READ.THE.MENU (figure 4). Routine INITIALIZE (figure 5) sets up the starting conditions for the simulation and finally, the graphics portion is initialized in INITIALIZE.GRAPHICS (figure 6).

Graphics will be discussed as a separate topic. For now, let us concentrate on the simulation aspects of the model.

Routine READ.THE.MENU (figure 4) illustrates a means of displaying the current parameters and cycling through them allowing the user to change as few or as many as he/she wishes. Two possible exits from this cycle are either to "Run" the simulation or "Exit", which will terminate the program without running the simulation. The PRINT statements are followed by their formats which are copied verbatim to the output device (screen). Any variables to be printed are placed where asterisks appear in the format. WRITE statements are similar to PRINT statements but are used to allow the cursor to remain on the same line. The VGOTOXY library routine is used to position the cursor on the menu.

Routine INITIALIZE (figure 5) activates the initial processes for the simulation. (There must be at least one pending process when the simulation starts.) Two copies of LOAD.GENERATOR are activated, one for each floor. Each potentially has a different inter-arrival rate passed to it as a parameter, and each has the floor number as the second parameter. The LIFT is activated immediately and the FINAL.REPORT is scheduled for occurrence after a delay of STOP.TIME hours.

The Simulation Routines

Process LOAD.GENERATOR (figure 6) models the arrival of new loads on each floor. When a new load arrives, it is placed in queue and if the lift is idle, it is reactivated. (Other code in this routine pertains to positioning the load for display.)

Process LIFT (figure 7) is the representation of the physical lift. It operates in a continuous loop moving between floors, examining LOAD.QUEUES, loading, and unloading. The code should be self-explanatory. The only additions to the logic for animation purposes are the setting of the velocity.a attribute and the LIFT.STATUS attribute. The velocity.a attribute controls the animation of the lift icon and LIFT.STATUS is used to control the color of the icon (red for stopped and green for moving).

Statistics and the Final Report

Process FINAL.REPORT (figure 8) pauses to wait for the user to finish admiring the animation (the READ AS / from UNIT 5) then proceeds to print the final text results from the model. The statistical results were all generated from the ACCUMULATE and TALLY statements which appear in the preamble. These statements declare what results are required but not how to collect the data. SIMSCRIPT II.5 automatically monitors the variables in the ACCUMULATE and TALLY statements (CONTENTS and n.LOAD.QUEUE in our example) and whenever their values change, the necessary statistical variables are updated.

Presentation Graphics

A variety of presentation graphics are available in SIMSCRIPT II.5. Several are used in our model. The queue length on each level may be displayed in several ways. The program includes options to display them as plots (queue length vs. time), level meters (much like a thermometer), or dials. In addition simulation time is displayed as either a 12 or 24 hour analog clock. These presentation icons were prepared using the PC SIMSCRIPT II.5 presentation graphics editor. This editor is completely menu driven. It is used to prepare presentation graphics before, during, or after model development. The icons may be changed at any time without any need to recode the model.
A example of a presentation graphics edit screen is shown in figure 12.

Adding Animation

The entity/attribute/set structure of SIMSCRIPT II.5 extends very nicely to allow the inclusion of animation in a simulation model. Entities which are to be displayed are declared as GRAPHICAL ENTITIES and those which are also to move are declared as DYNAMIC GRAPHICAL ENTITIES. These declarations (in the preamble) cause the compiler to include additional attributes which are required for these purposes. The icons are usually prepared with the Icon Editor and associated with their respective entities with the DISPLAY statement.

Routine INITIALIZE.GRAPHICS (figure 9) initializes a "virtual terminal" to display the graphical output of our model. Several objects are drawn as part of the background. These appear in figure 14 as the ground, a building and a truck. These icons were constructed with the Icon Editor and may be changed at will. The lift is displayed with a user-written display routine. The association of this routine with the process is made in the initialization program. The "smart icons" for queue and clock displays are initiated here as well.

Routine UPDATE.CLOCK (figure 10) forces the displayed clock to keep up with simulated time. This routine can be enhanced to do such things as "time warp", i.e., leap ahead when nothing is moving on the screen. The usual convention is to produce an audible signal when such a leap occurs.

The display routine for the lift (figure 11) had to be written by the user rather than to use a system-generated default routine because of the need to change the color of the lift based on its STATUS attribute and the desire to redraw all the load icons which represent loads within the lift. This is more efficient than treating the loads as dynamic entities in their own right and avoids the "rubber band" effect of moving the lift and then moving each load on the lift.

SIMSCRIPT II.5 AVAILABILITY

SIMSCRIPT II.5 is the proprietary product of CACI International Inc. It is sold on a free-trial basis.

A special university program is supported by CACI in which SIMSCRIPT II.5 is supplied to educational institutions for the cost of distribution.

TRAINING

Week-long training courses are given by CACI on a regular basis. These courses are held in their training facilities in Los Angeles and Washington, D.C., as well as at other locations throughout the world. The same course is available for on-site training as well. For further information on courses, contact:

CACI International Inc
3344 N. Torrey Pines Court
La Jolla, CA 92037
Phone: (619) 457-9681

```
preamble
   normally mode is undefined

   processes include FINAL.REPORT
      every LOAD.GENERATOR
         has a MEAN.RATE
         and a LOAD.FLOOR
      define MEAN.RATE as a real variable
      define LOAD.FLOOR as an integer variable

      every LIFT
         has a LIFT.STATUS
         and owns a LIFT.LOAD
      define LIFT.STATUS as a text variable

   permanent entities
      every LEVEL
         has a QUEUE.ICON
         and owns a LOAD.QUEUE
      define QUEUE.ICON as a text variable

   temporary entities
      every LOAD
         has a FLOOR
         and may belong to the LOAD.QUEUE
         and may belong to the LIFT.LOAD
      define FLOOR as an integer variable

   define LIFT.CAPACITY as an integer variable
   define STOP.TIME and CONTENTS as real variables
   define LIFT.WAIT as an integer variable

   tally AVG.CONTENTS as the average
+      and NO.OF.LOADS as the number of CONTENTS
   accumulate AVG.QUEUE as the average
      and MAX.QUEUE as the maximum of n.LOAD.QUEUE

   define MINUTES to mean units
   define HOURS to mean * 60.0 units

   define LOAD.QUEUE as a LIFO set
                     '' cheat ! ..no  need to re-draw all
                     '' loads as they move up 1 place

'' SimAnimation graphic declarations

   graphic entities include LOAD        '' these are visible
   dynamic graphic entities include LIFT ''..and this moves!

   graphic entities include SHAPE

   define LOADSHAPE as a pointer variable

   define SCALER as a real variable  '' time scaling factor

   define .STOPPED.COLOR to mean 2   '' red
   define .MOVING.COLOR to mean 3   '' green

   define CLOCKTIME as a double variable
   define TIMEICON as a text variable
   display variables include n.LOAD.QUEUE, CLOCKTIME

end  ''preamble
```

Figure 1 - The PREAMBLE

```
main
   call SET.DEFAULTS
   call READ.THE.MENU
   call INITIALIZE
   call INITIALIZE.GRAPHICS
   start simulation
end  '' main
```

Figure 2 - MAIN

```
routine to SET.DEFAULTS
   create every LEVEL(2)
   let LIFT.CAPACITY = 3
   let STOP.TIME = 12  ''hours
   let SCALER = 30
   let TIMEICON = "aclock"
   let QUEUE.ICON(1) = "trace1"
   let QUEUE.ICON(2) = "trace2"
end  '' routine to SET.DEFAULTS
```

Figure 3 - Routine SET.DEFAULTS

```
routine to READ.THE.MENU
   define ICON.NAME as a text variable
   define INPUT.TIME as an integer variable
   define CHOICE as an alpha variable
   define DONE as a text variable
   let DONE = "n"
   until DONE = "y"
   do
      call vclears.r
      print 8 lines with STOP.TIME,TIMEICON,SCALER,
         QUEUE.ICON(1),QUEUE.ICON(2) thus

             >>>>>>>>>> GOLDMINE SIMULATION MENU <<<<<<<<<<

                              Default values
Simulated run time is *** hours    Time icon = **********
Time scale is *** real seconds per simulated hour
                               Queue 1 icon = ***********
                               Queue 2 icon = ***********

      print 11 lines thus

 1) Change the simulated run length (hrs.)
 2) Change the time scale (sec/hr)

 3) Change the icon for time      (available list of icons
 4) Change the icon for queue 1    is displayed when you
 5) Change the icon for queue 2    select the menu item.)

 R) Run the simulation
 E) Exit the simulation

      call vgotoxy.r(21,0)
      write as "Enter your choice => ",+
      call rcr.r
      read CHOICE as A 1
      call vgotoxy.r(21,0)
      call vclearl.r

      select case CHOICE
      case "1"
         write as "Enter new simulated run time in hours =>"
            ,+
         read INPUT.TIME
         if INPUT.TIME > 0 and INPUT.TIME <= 99999
            let STOP.TIME = INPUT.TIME
         always

      case "2"
         write as "Enter new time scale",/,
               "(real seconds per simulated hour => ",+
         read INPUT.TIME
         if INPUT.TIME > 0 and INPUT.TIME <= 3600
            let SCALER = INPUT.TIME
         always

      case "3"
         write as "Enter new icon for the clock",/,
               "     (aclock or 24clock) => ",+
         read ICON.NAME
         let TIMEICON = ICON.NAME

      case "4"
         write as "Enter new icon for queue 1",/,
               "(trace1, level1, or dial1) => ",+
         read ICON.NAME
         let QUEUE.ICON(1) = ICON.NAME

      case "5"
         write as "Enter new icon for queue 2",/,
               "(trace2, level2, or dial2) => ",+
         read ICON.NAME
         let QUEUE.ICON(2) = ICON.NAME

      case "R", "r"
         let DONE = "y"
      case "E", "e", "X", "x"
         stop

      default
      endselect
   loop
end '' routine to READ.THE.MENU
```

Figure 4 - Routine READ.THE.MENU

```
routine INITIALIZE
   activate a LOAD.GENERATOR giving 1 / 5 and 1 now
   activate a LOAD.GENERATOR giving 1 / 5 and 2 now
   activate a LIFT now
   let STOP.TIME = STOP.TIME * 60
   activate a FINAL.REPORT in STOP.TIME minutes
end '' routine INITIALIZE
```

Figure 5 - routine INITIALIZE

```
process LOAD.GENERATOR given INTER.ARRIVAL.TIME and
LOAD.FLOOR
   define INTER.ARRIVAL.TIME as a real variable
   define LOAD.FLOOR as an integer variable
   define X and Y as real variables
   until time.v >= STOP.TIME
   do
      wait exponential.f(INTER.ARRIVAL.TIME,1) HOURS
      create a LOAD
      let FLOOR(LOAD) = LOAD.FLOOR

''    give the load an icon and position it at end of queue
      let X = 15 + n.LOAD.QUEUE(LOAD.FLOOR) * 12.0
      let Y = -100.0 * (3-LOAD.FLOOR)
      display LOAD with "LOAD" at (X, Y)
      file LOAD in LOAD.QUEUE(LOAD.FLOOR)
      if LOAD.FLOOR = 1
         and sta.a(LIFT) = 2 '' ie, the lift is idle
         reactivate the LIFT now
      always
   loop
end '' process LOAD.GENERATOR
```

Figure 6 - Process LOAD.GENERATOR

```
process LIFT
   define LIFTSPEED as a real variable

   let LIFTSPEED = 100.0 / 3.0        '' ft/min

   until time.v >= STOP.TIME
   do
      let velocity.a(LIFT) = velocity.f(LIFTSPEED, -PI.C/2)
      let LIFT.STATUS(LIFT) = "moving"
      wait 3 MINUTES  ''to descend to level 2

      if LOAD.QUEUE(1) is not empty
         or LOAD.QUEUE(2) is empty,
         wait 3 MINUTES  ''to descend to level 1

         let velocity.a(LIFT) = 0
         let LIFT.STATUS(LIFT) = "stopped"
         if LOAD.QUEUE(1) is empty,
            suspend
         always

         until n.LIFT.LOAD = LIFT.CAPACITY
            or LOAD.QUEUE(1) is empty
         do
            remove the first LOAD from LOAD.QUEUE(1)
            let location.a(LOAD) = 0           disappear
            file this LOAD in LIFT.LOAD
            display LIFT
            wait uniform.f(0.75, 1.25, 1) MINUTES
         loop

         let velocity.a(LIFT) = velocity.f(LIFTSPEED,
                                           PI.C/2)
         let LIFT.STATUS(LIFT) = "moving"
         wait 3 MINUTES ''to ascend to level 2

      always

      if LOAD.QUEUE(2) is not empty
         and n.LIFT.LOAD < LIFT.CAPACITY,
         let velocity.a(LIFT) = 0
         let LIFT.STATUS(LIFT) = "stopped"
         display LIFT
         wait 0.5 MINUTES
         until n.LIFT.LOAD = LIFT.CAPACITY
            or LOAD.QUEUE(2) is empty,
         do
            remove the first LOAD from LOAD.QUEUE(2)
            let location.a(LOAD) = 0           '' disappear
            file this LOAD in LIFT.LOAD
            display LIFT
            wait uniform.f(0.75, 1.25, 1) MINUTES
        loop
        display LIFT
        wait 0.5 MINUTES
        let velocity.a(LIFT) = velocity.f(LIFTSPEED, PI.C/2)
        let LIFT.STATUS(LIFT) = "moving"

      always

      wait 3 MINUTES ''to ascend to surface

      let velocity.a(LIFT) = 0
      let LIFT.STATUS(LIFT) = "stopped"

      let CONTENTS = n.LIFT.LOAD
      until LIFT.LOAD is empty,
      do
         remove the first LOAD from LIFT.LOAD
         destroy this LOAD
         display LIFT
         wait uniform.f(0.75, 1.25, 1) MINUTES
      loop

   loop
end  ''process LIFT
```

Figure 7 - Process LIFT

```
process FINAL.REPORT
   use 5 for input
   read as /
   use 6 for output
   print 13 lines with time.v, AVG.CONTENTS, NO.OF.LOADS,
      AVG.QUEUE(1), MAX.QUEUE(1), AVG.QUEUE(2), MAX.QUEUE(2)
thus

   Results after ********.**** simulated minutes

      Average lift contents were ***.******
      Number of lift trips was  ******

      Average queue at the first level was  ***.****
      Maximum queue at the first level was  ******

      Average queue at the second level was ***.****
      Maximum queue at the second level was ******


   stop
end  ''process FINAL.REPORT
```

Figure 8 - Process FINAL.REPORT

```
routine INITIALIZE.GRAPHICS
   define DEVICE.ID as a pointer variable
   define GROUND, BUILDING and TRUCK as pointer variables
   let timescale.v = SCALER * 100 / 60.0
                        ''clock ticks (1/100 sec) /  unit
   let timesync.v = 'UPDATE.CLOCK'

'' Create a 'Graphic display' using 2 new I/O units

   call devinit.r("VT,GRAPHIC") yielding DEVICE.ID
   open 7 for input, device = DEVICE.ID
   open 8 for output, device = DEVICE.ID
   use 8 for graphic output

'' Select a viewing transform and establish its mapping

   let vxform.v = 1
   call setworld.r(-50.0, 200.0, -220.0, 20.0)

   create a SHAPE called GROUND
   display GROUND with "ground"
   create a SHAPE called BUILDING
   display BUILDING with "build" at (80, 0)

   create a SHAPE called TRUCK
   display TRUCK with "truck" at (150, 0)

'' override the standard routine
   let drtn.a(LIFT) = 'V.LIFT'
   let LIFT.STATUS(LIFT) = "stopped"
   display LIFT with "lift" at (0, 0)

   create a SHAPE called LOADSHAPE
   show LOADSHAPE with "load"

   let vxform.v = 0

   display CLOCKTIME with TIMEICON

   display n.LOAD.QUEUE(1) with QUEUE.ICON(1)
   display n.LOAD.QUEUE(2) with QUEUE.ICON(2)

   let vxform.v = 1
end ''routine INITIALIZE.GRAPHICS
```

Figure 9 - routine INITIALIZE.GRAPHICS

```
routine UPDATE.CLOCK given TIME yielding NEWTIME
   define TIME, NEWTIME as double variables

   let NEWTIME = TIME
   let CLOCKTIME = TIME / (60 * 24)

   return
end '' routine UPDATE.CLOCK
```

Figure 10 - Routine UPDATE.CLOCK

```
display routine LIFT(LIFT)

  define LIFT  and LOAD as pointer variables

   let vxform.v = 1  '' set viewing transform
   if LIFT.STATUS(LIFT) =  "stopped"
      call fillcolor.r(.STOPPED.COLOR)
   else
      call fillcolor.r(.MOVING.COLOR)
   always

   display icon.a(LIFT)
   call mxlate.r(-12.0, 0.0)
   for each LOAD in LIFT.LOAD(LIFT)
   do
      display icon.a(LOAD)
      call mxlate.r( 8.0, 3.0)
   loop

end ''display routine LIFT
```

Figure 11 - Display Routine LIFT

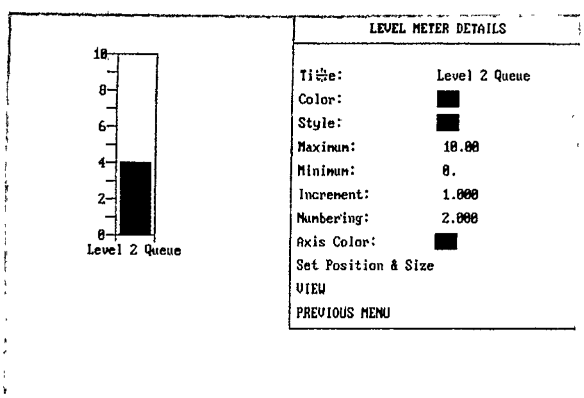

Figure 12 - Preparation of a Presentation Graphic

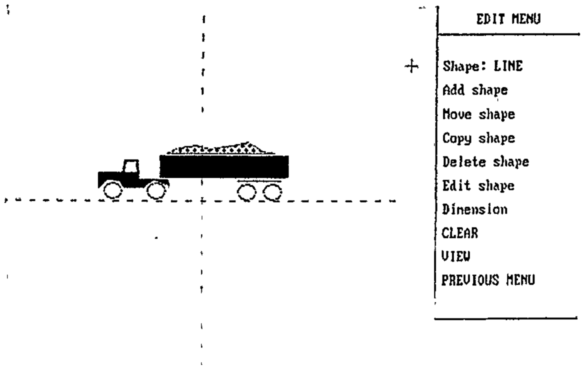

Figure 13 - Preparation of an Animation Icon

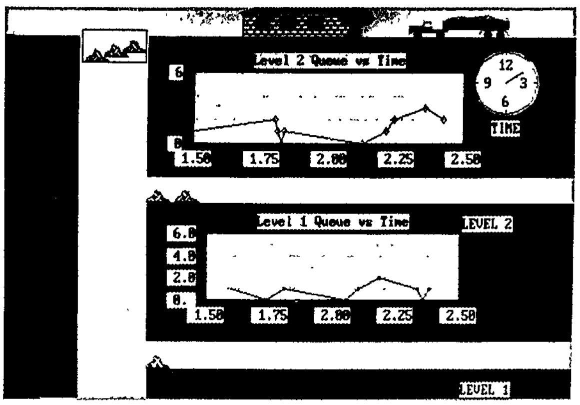

Figure 14 - A Snapshot of the Animated Simulation

```
                    >>>>>>>>>> GOLDMINE SIMULATION MENU
<<<<<<<<<<

                                        Default values
Simulated run time is  12 hours               Time icon =
aclock
Time scale is  30 real seconds per simulated hour
                                           Queue 1 icon =
trace1
                                           Queue 2 icon =
trace2


     1) Change the simulated run length (hrs.)
     2) Change the time scale (sec/hr)

     3) Change the icon for time        (available list of
icons
     4) Change the icon for queue 1      is displayed when
you
     5) Change the icon for queue 2      select the menu
item.)

     R) Run the simulation
     E) Exit the simulation

     Enter your choice => r
```

Figure 15 - The Opening Menu

```
  Results after       720.0000 simulated minutes

     Average lift contents were    2.813953
     Number of lift trips was       43

     Average queue at the first level was      .8380
     Maximum queue at the first level was        4

     Average queue at the second level was    2.6098
     Maximum queue at the second level was       9
```

Figure 16 - A Sample of the Final Results

SIMSCRIPT II.5 Bibliography

The following publications are available from CACI:

Building Simulation Models with SIMSCRIPT II.5 by Edward C. Russell, 1983.

Introduction to Simulation Using SIMSCRIPT II.5 by Averill M. Law and Christopher S. Larmey, University of Arizona, Tempe, 1984.

SIMSCRIPT II.5 Programming Language by Harry M. Markowitz, P.J. Kiviat and R. Villanueva (1987 edition).

SIMSCRIPT II.5 Reference Handbook, 1987 edition.

PC SimAnimation User's Guide and Case Book, 1987 edition.

Introduction to Combined Discrete-Continuous Simulation Using PC SIMSCRIPT II.5 by Abdel-Moaty M. Fayek, California State University, Chico, 1987

PC SIMSCRIPT II.5 Introduction and User's Manual, 1987 edition.

and User's Manuals for all other implementations

AUTHOR'S BIOGRAPHY

EDWARD C. RUSSELL is Vice President of CACI International Inc. and manager of the Modelling and Simulation Depart-ment. He has responsibility for software products including SIMSCRIPT II.5, NETWORK II.5, and SIMFACTORY. He has developed and regularly teaches a one-week short course on Simulation and SIMSCRIPT II.5. He is also actively engaged in major modelling and simulation projects. He is the author of the teaching text for SIMSCRIPT II., ***Building Simulation Models with SIMSCRIPT II.5***. He is also an adjunct professor in the Graduate Schools of Computer Science and Management at the University of California at Los Angeles (UCLA). Dr. Russell's academic background includes a BSEE from Wayne State University, and MSE and PhD degrees in computer science from UCLA.

Edward C. Russell
CACI International Inc.
12011 San Vicente Blvd.
Los Angeles, CA 90049
(213) 476-6511