Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

SIMULATION IN PASCAL WITH MICRO PASSIM

Claude C. Barnett
Physics Department
Walla Walla College
College Place, WA 99324, U.S.A.

ABSTRACT

The micro PASSIM package is designed to support combined simulation modeling in Pascal, including the coordinated sequencing of discrete events and the integration of continuous variables. The package features dynamic interaction through user and system menus, resource and queue management, random number streams and generation of statistical distributions, the collection of statistics, report generation, tracing of transactions, dynamic model interaction and character graphics display of model evolution. Event scheduling and process interaction world views are supported. The modeler writes the model in Pascal using the procedures from the simulation library for support. The model and support procedures are compiled together to provide a stand-alone simulation program. Version mP 3.1 using Turbo Pascal supports high resolution graphics. Complete source code is provided for either Turbo or Vax Pascal. Version mp 3.1 M2 in Modula-2 is now available.

1. INTRODUCTION TO MICRO PASSIM

The decision to implement micro PASSIM in the high level language Pascal is based on the premise that a complete simulation language will have all the structures of a good high level language. The argument to support this premise begins with the observation that the same expressive power is needed to program a model as to write a program in general.

The high level language does need a supporting environment to organize model construction, facilitate model verification and experimentation and generally reduce the work of modeling by supplying frequently used simulation constructs. micro PASSIM is designed to fill this need. The simulation library provides the special support needed for simulation, while Pascal ensures power of expression and extensibility. Modelers need a simulation system that allows choice of different world views and simulation metaphors. Pritsker (1984) has recognized the importance of simulation languages with the capability of alternative modeling strategies. Zeigler (1984) has presented a cogent case for multifaceted modeling and simulation.

PASSIM was first coded by Uyeno (1980) in Pascal to run on a main frame and adapted by Barnett (1981) to run on a microcomputer as micro PASSIM. The original versions were GPSS-like, supporting transaction flow. Now versions of micro PASSIM have been extended by Barnett (1983, 1985) and Anderson (1986) to allow discrete-event scheduling, network modeling, and continuous state variable integration as well as process interaction.

An example model introduces some of the features of micro PASSIM and illustrates a degree of multifaceted modeling.

2. EXAMPLE MODEL

Model construction is only one phase of an iterative process leading to a successful simulation study (Balci, 1985). It is vital that the method of model implementation make provisions for experimentation that lead to specific results that address the formulated problem. The development of a simple model will illustrate the features of micro PASSIM that support such a simulation study.

The micro PASSIM package contains a model "template" that serves as a guide to model construction. Iterative entries into the seven main segments of this template will lead to a working model. The seven segments are designed to emphasize important aspects of model design: (1) describe the model, (2) identify and declare the model entities and their attributes, (3) specify typical values of attributes of entities, (4) select entities and attributes to be changed during the model experimentation phase, (5) specify the initial configuration of the system, (6) specify reports and statistical results, and (7) specify the model structure. These steps are to be done iteratively.

Each of the procedures described here are needed for system control of the simulation. The effort to construct the model will be rewarded by a program suited to interactive experimentation.

2.1. Model Description

A carefully worded description of the system being modeled is helpful to both the modeler and to the model user. Instructions to the user can also be provided here. This documentation is accessible from the main menu. In the example model we include in MODEL_DESCRIPTION statements of the form "WRITELN(OUT,'...text...');", or a file called by the library procedure SHOW_FILE. A communicated model describing the Bobwhite Quail life cycle is used as an example:

"Quail in a region increase in number in the spring during mating season, die from natural causes at a rate that changes with the seasons, and are hunted in season. The quail population growth rate equals the product of the current population and the difference between the birth rate and the death rate. The death rate equals the sum of the natural death rate and the death rate due to hunting minus the product of the two. The latter product is the compensatory effect of hunting quail that might have died anyway. The natural death rate is a function of the ratio of the current quail population to the carrying capacity of the region. The birth rate also depends on this ratio. The problem is to determine which wildlife management policies will ensure a stable quail population."

Programing of a part of this communicated model will now be described as a tutorial.

2.2. Model Entities

An early step in model conceptualization is to identify the system and the dynamic and static entities of the system together with their attributes. The data structures available for modeling will influence this choice. The activities engaged in by the entities and the existence of causal relationships will also influence the representation of entities. In our example we choose to model the quail population with a continuous state variable:

```
VAR QUAIL : STATE_PTR;
```

This choice is influenced by the observed causal relationship between quail population and birth and death rates. The plan is to describe this causal relationship by a differential equation, which will be integrated by micro PASSIM.

The hunters will be represented as a state variable, primarily to make it easy to plot their number. Similarly, the number of hunter kills, the bag, will be represented as a state variable:

```
VAR HUNTERS : STATE_PTR;
    BAG     : STATE_PTR;
```

Pascal pointers allow access to each record field containing attributes of the state variable, such as initial and current levels, the rate of change of the levels, and plotting information. For example the quail level would be QUAIL^.LEVEL.

The hunt will be represented by one of the two main resources of micro PASSIM:

```
VAR HUNT : STORE_PTR;
```

This choice is made to allow the hunters to be modeled as discrete events, knowing that the micro PASSIM library contains several procedures for managing the STORE resource. For each STORE, the system creates a CHAIN, the other micro PASSIM resource, to model waiting for the resources of the STORE, active hunters, to become available.

The HUNTERS will be updated as the HUNT progresses by statements of the form:

```
HUNTERS^.LEVEL := HUNT^.USE;
```

The birth and death rates change with the seasons of the year, and may be represented by empirical data tables expressing the fertility and mortality factors as functions of time of year:

```
VAR FERTILITY : TABLE_PTR;
    MORTALITY : TABLE_PTR;
```

A number of other model parameters may be declared as REAL, INTEGER, BOOLEAN, CHAR, or any of the other Pascal types:

```
VAR QUAIL_MAX : REAL; {carrying capacity}
    QUAIL_FALL: REAL; {fall population   }
```

Model entities, attributes and activities may also be represented by functions or procedures the user may choose to write in Pascal. All of the declared variables will be given initial or "default" values as a modeling step.

2.3. Model Defaults

Pascal does not initialize any variables declared by the user. Although micro PASSIM does take care of as much initialization as practical, it is an important part of the modeling process to choose typical starting values for model variables. The MODEL_MENU will allow changing any selected variables during a model run.

Within the Bobwhite Quail example the QUAIL record is created by the micro PASSIM procedure NEW_STATE and initialized to a level of 3000 with the name 'quail'. Plot parameters are set for QUAIL by SHOW_STATE to choose a high resolution plot mode, to set the data window between 0 and 6000 quail, to set the y-axis data port between 180 and 20 screen units, and to set the plotting offset to 0. All of these values can be changed at run time from the plot menu:

```
QUAIL := NEW_STATE('quail', 3000);
SHOW_STATE(QUAIL,CHR(1),0,6000,180,20,0);

QUAIL_MAX := 1.75*3000;
```

The carrying capacity for the region is set to an experimentally determined factor above the fall population. The carrying capacity can be changed if it is included in the procedure MODEL_MENU, to be written by the modeler.

2.4. Model Menu

An important part of any simulation study is experimentation. The micro PASSIM menus allow system and user defined properties to be changed at run time. The MODEL_MENU is specifically designed to make it easy to change variables defined by the user, and not known to the system. As an example, menu items for the hunting season, the average number of hunters per day and the daily bag would be coded as follows:

```
PROCEDURE MODEL_MENU;
BEGIN CLR_SCREEN(OUTPUT); PREVIEW;
REPEAT
ACCEPT_R( 4, 'open hunting season day     '
            , 0, 365, HUNTING_OPEN, DEC_PL);
ACCEPT_R( 5, 'length of hunting season    '
            , 0, 365, SEASON_H,     DEC_PL);
ACCEPT_R( 6, 'average number of hunters '
            , 0, 5000, HUNTERS_AVE, DEC_PL);
ACCEPT_R( 7, 'daily bag per hunter        '
            , 0, 100,  DAILY_BAG,   DEC_PL);
UNTIL PREVIEWED;
BAG_RATE    := (DAILY_BAG/DAY)/QUAIL^.LEVEL0;
BAG_INDEX   := BAG_RATE*HUNTERS_AVE;
HUNTERS_MAX := 2*ROUND(HUNTERS_AVE);
END; (* MODEL_MENU *)
```

2.5. Model Initialization

For each new scenario the model needs to be initialized. The procedures MODEL_INIT and MODEL_RESET do this.

The HUNT resource is created and set to initial conditions by a micro PASSIM library procedure called in MODEL_INIT:

```
  HUNT := NEW_STORE('hunt',200,TRUE);
```

The maximum amount of the HUNT resource allowed, number of active hunters, is set to 200. The TRUE signifies that statistics will be kept for this STORE. The user has access to the fields of the created resource from a micro PASSIM menu at run time.

The system resets the statistics on any STORE created by NEW_STORE and on any CHAIN created by NEW_CHAIN. The user may use the MODEL_RESET procedure to reset any special statistical variables. 'RESET' is available from the main menu, and is called by the system at 'CLEAR' time, also selectable from the main menu.

2.6. Model Reports

Statistical reports will be generated by the system on resources for which statistics were kept. These may be selected from the main menu, and redirected to a file or to a printer.

In addition to the system generated reports on resources, other information that would be of value for our model would be the fall quail population, the total number of quail taken in hunting, and the total number of hunters.

The character graphics and the high resolution plot of the state variables gives a dynamic picture of system evolution. The screen may be printed periodically as part of the record of system performance.

2.7. Model Structure

The activities of the model entities are captured and expressed in the procedure MODEL. This procedure controls calls to the procedure DERIVATIVES where the model's causal relationships are expressed as the time rate of change of the levels of the state variables.

The main factors affecting the growth of the Bobwhite population are expressed in the following segment of DERIVATIVES:

```
PROCEDURE DERIVATIVES;
CONST
 DF = 1.240;       {empirical density  factor}
 CF = 0.940;       {empirical carrying factor}

VAR
 NaturalDR, BagDR, RelPop:REAL;

BEGIN
RelPop        := QUAIL^.LEVEL/QUAIL_MAX;
NaturalDR     := INTERPOLATE(MORTALITY, CLOCK)
      *(1 + DF*RelPop - CF*SQR(1 - RelPop));
BagDR         := BAG_RATE*HUNTERS^.LEVEL;
QUAIL^.RATE :=  - QUAIL^.LEVEL*
      *(NaturalDR + BagDR - NaturalDR*BagDR);
IF MATING
   THEN QUAIL^.RATE := QUAIL^.RATE
              + INTERPOLATE(FERTILITY, CLOCK)
              * QUAIL^.LEVEL;
IF HUNTING
   THEN BAG^.RATE := BagDR*LEVEL;
END; (* DERIVATIVES *)
```

The PRIME node of MODEL is called by the system to start the simulation, processing discrete events first. In our example this node is used to start the timing segments that will control the yearly seasons, the mating season, and the hunting season.

```
PROCEDURE MODEL;
BEGIN IF DISCRETE THEN
               WITH CUR^ DO CASE NX_BLOCK OF

PRIME:BEGIN                (*>> required <<*)
  SEASON  := SUMMER;
  SCHEDULE (WINTO, TOP_PRIORITY,
            WINTER_START, NEW_XACT(CUR));
  HUNTING := FALSE;
  SCHEDULE (HUNTO, TOP_PRIORITY,
            HUNTING_OPEN, NEW_XACT(CUR));
  MATING  := FALSE;
  SCHEDULE (MATE0, TOP_PRIORITY,
                     MATING_START, CUR);
  DAILY_BAG := BAG_INDEX*DAY
                    *QUAIL0/HUNTERS_AVE
  END;
```

In the hunting cycle the number of hunters hunting on a given day is chosen randomly from a triangular distribution using random number stream 1. The mean of the distribution is equal to the previously specified average number of hunters.

```
HUNTO:BEGIN
  IF HUNTING
    THEN HUNTERS^.LEVEL := TRIANGLE(0,
         HUNTERS_AVE, 2*HUNTERS_AVE, STREAM1)
    ELSE HUNTERS^.LEVEL := 0;
  ENTER(HUNT1, ROUND(HUNTERS^.LEVEL), HUNT)
  END;
HUNT1:ADVANCE(HUNT2,DAY);
HUNT2:BEGIN              (* end of daily hunt *)
  HUNTERS_SUM := HUNTERS_SUM + HUNT^.use;
  HUNTERS^.LEVEL := 0;
  LEAVE(HUNT3, HUNT^.use, HUNT)
  END;
HUNT3:ADVANCE(HUNT4, 1 - DAY);
HUNT4:TRANSFER(HUNT5, HUNT1, TRUE, HUNTING);
HUNT5:TERMINATE(1);  (* count hunting days *)
```

In the hunting season segment the fall quail level is saved and the relative quail harvest is written to OUT, the redirectable output buffer. The procedures ENABLE and DISABLE act on the HUNT STORE to enable and disable use of the STORE by hunters.

```
HUNS0:BEGIN      (* hunting timing segment *)
  HUNTING := TRUE;
  QUAIL_FALL := QUAIL^.LEVEL;
  ENABLE  (HUNS1, HUNT) END;
HUNS1:ADVANCE (HUNS2, SEASON_H);
HUNS2:BEGIN
  HUNTING := FALSE;
  GOTOXY( 28, 2);
  WRITELN(OUT, 'BAG = '
              , BAG^.LEVEL/QUAIL_FALL:5:3);
  BAG_SUM := BAG_SUM + BAG^.LEVEL;
  QUAIL_SUM := QUAIL_SUM + QUAIL_FALL;
  BAG^.RATE := 0; BAG^.LEVEL := 0;
  DISABLE(HUNS3, HUNT) END;
HUNS3:ADVANCE (HUNS0, YEAR - SEASON_H);
```

The mating timing segment sets the boolean variable MATING true during the mating season and false otherwise. This information is used in setting the level of the fertility variable.

```
MATE0:BEGIN      (* mating timing segment *)
      MATING := TRUE;
      ADVANCE (MATE1, SEASON_M) END;
MATE1:BEGIN
      MATING := FALSE;
      ADVANCE (MATE0, YEAR - SEASON_M) END;
```

The seasons are controlled by the following timing segment:

```
WINT0:BEGIN      (* winter timing segment *)
      SEASON := WINTER;
      ADVANCE (WINT1, SEASON_W) END;
WINT1:BEGIN
      SEASON := SUMMER;
      ADVANCE (WINT0, YEAR - SEASON_W) END;
```

Reports can be generated at periodic intervals set from the main menu. These reports can be sent to the screen, a file, or the printer.

```
REPTS:IF REP_INT<=0         (*>> required <<*)
 THEN TERMINATE(0)
 ELSE BEGIN
   ADVANCE (REPTS, REP_INT);
   IF DISPLAY THEN PLOT_ALL([STATES])
   END;

 STOP:HALT_SIMULATION;      (*>> required <<*)

 END (* CASE *)
```

The integration of the state variables is controlled by this last part of the MODEL procedure. The sequencer and integrator procedures control the advance of time either by discrete steps when the discrete part of the model is active or by user specified uniform time steps between events when the continuous part of the model is active.

```
ELSE IF DE_ON               (*>> required <<*)
 THEN REPEAT
   INTEGRATE; DERIVATIVES
 UNTIL TIME_OUT END;     (* MODEL *)
```

The general part of the code for MODEL is already contained in the TEMPLATE file supplied with the system. The modeler adds the model specific code to the TEMPLATE to produce the final model.

3. CONCLUSIONS

The micro PASSIM simulation package gives the model builder a supportive environment for combined discrete-event and continuous state variable simulation. The simulation library helps organize and simplify the implementation of modeling concepts, allows a multifaceted approach to simulation, and provides for model experimentation through model interaction.

4. AVAILABILITY

The micro PASSIM simulation package is currently available in version mP 3.10 for PC compatibles using Turbo Pascal version 2.0+ and for the VAX using VMS 3.0+ and DEC Pascal 2.0+. Full source code and users manual with documentation is supplied with each version. The compiled library and system commands are supplied with the VAX version. The Modula-2 version and version mP 4.00 used in this tutorial are available for field testing.

ACKNOWLEDGMENTS

Work on the example quail model was done at the University of Arkansas Department of Zoology for the Arkansas Game and Fish Commission by David J. Stewart, Charles J. Amlaner, Jr. and Claude C. Barnett.

REFERENCES

Anderson, T. L., Barnett, C. C. (1986). Modula-2 With An Enriched Library As A Simulation Environment: micro PASSIM M2, Modeling and Simulation on Microcomputers, Claude C. Barnett, Editor, Society for Computer Simulation, San Diego, pp240-242.

Balci, Osman (1985). Guidelines for Successful Simulation Studies, Report Number TR-85-2, Naval Sea Systems Command, Office of Naval Research.

Barnett, Claude C.(1981). micro PASSIM, A Discrete-Event Simulation Package For A Microcomputer Using UCSD Pascal, Modeling and Simulation on Microcomputers, L. A. Leventhal, Editor, Society for Computer Simulation, La Jolla, pp 60-64.

Barnett, Claude C.(1983). micro PASSIM: A Combined Simulation Package For A Microcomputer Using UCSD Pascal, Modeling and Simulation on Microcomputers, Ralph Martenez, Editor, Society for Computer Simulation, La Jolla, pp 92-95.

Barnett, Claude C.(1985). micro PASSIM: A Modeling Package for Combined Simulation Using Turbo Pascal, Modeling and Simulation on Microcomputers, Greer Lavery, Editor, Society for Computer Simulation, La Jolla, pp 37-41.

Pritsker, A.A.B. (1984). Introduction to Simulation and SLAM II. Second Edition, Halstead Press, New York.

Uyeno, Dean (1980). PASSIM, A Discrete-Event Simulation Package for Pascal, Simulation, Vol 35, No. 6, pp 479.

Zeigler, B.P. (1984). Multifacetted Modeling and Discrete Event Simulation. Academic Press, London, England.

AUTHORS' BIOGRAPHIES

CLAUDE C. BARNETT is a professor of physics in the Department of Physics at Walla Walla College. He received a B.S. degree in physics from Walla Walla College in 1952 and M.S. and Ph.D. degrees in physics from Washington State University in 1956 and 1960 respectively. He has been teaching modeling and simulation classes since 1968 and is the author of the micro PASSIM simulation package for microcomputers. His simulation related interests are in physical and biological models, simulation methodologies, languages, and information theory. He is a member of A.A.P.T., A.P.S., O.R.S.A., S.C.S., and the Society for Sigma Xi.

Claude C. Barnett
Department of Physics
Walla Walla College
College Place, WA 99324, U.S.A.
(509) 527-2881