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

THE SCRIPT PROCESSING TECHNIQUE IN MODELING/SIMULATION AND ITS ROLE IN THE GENERATION OF ANIMATED COMPUTER GRAPHICS

Robert E. Sheridan

The BDM Corporation

Albuquerque, New Mexico

ABSTRACT

The script processing method provides the modeler with an extremely powerful technique for developing simulations of force-on-force engagements that are detailed and/or complex in nature. Through the use of the script processing technique, it is possible to simulate the actions and reactions of opposing forces (logically) in parallel on a computer with a single CPU. The cornerstone of this technique is the input script and associated processing software. Many different scenarios and tactics can be played easily due to the flexibility of the script concept. Just as a script directs players within a framework of scenes and activities for a screenplay, the simulation script directs elements in particular actions such as movement, weapon firing, et cetera. It also provides for contingencies based on the recognition of script-defined situations.

The script concept lends itself as a natural driver to control the production of computer generated motion pictures. The discrete script events dictate the plot of the animated film and describe the movements and interaction between the picture elements. By algorithmic interpolation between script events, the discrete script format can be filled in to simulate the analog film medium. The Transportation Safeguards Effectiveness Model (TSEM) is a good example of the use of the script processing technique. This program will be used to demonstrate the above mentioned techniques in simulation and animated motion picture generation.

The graphics can also serve to guide the creation of macro/micro terrain features for the simulation scenario. In this way, the modeler has control over the placement of obstacles to both movement and line-of-sight calculations, for example. Also, the discrete script generated event vectors can be created off-line, a movie produced and viewed, and decisions made concerning the viability of this particular script. These capabilities enhancé the instruction and training of defender force configurations.

BACKGROUND

Script processing is a modeling technique that provides for the processing of player interactions by means of a special input structure and a parallel processing simulator (that is input/event driven). The script processing method provides the modeler with an extremely powerful approach for developing simulations of detailed/complex small force-on-force engagements. It provides a story input format (the input script). In addition, script processing simulates processes that normally occur simultaneously. Lastly, the script concept when implemented with an event-driven architecture lends itself as a natural driver to control the production of computer generated graphics.

The main basis for the script processing technique is provided by the script. The script is the input medium for defining the engagement scenario. The script directs elements (players, vehicles) in particular actions such as movement, weapon firing, and tactics, and it also provides for player assignments. In a real sense,

the simulation script resembles a screenplay in that the script directs players and objects within a framework of scenes and activities.

SCRIPT PROCESSING IMPLEMENTATION

The present implementation of the script processing methodology incorporates a major improvement that was designed and developed by the author and colleagues. This innovation consists of the addition of contingencies to the script and their associated processing. This feature provides the ability to make the performance of player/vehicle actions contingent on the occurrence of special events (for example, player killed, player reaches location, and so forth). Another significant enhancement is the provision of a mechanism (that is, the script input language and processor) for expressing the script in an English-like manner instead of a numerically encoded form. A detailed discussion of the present script processing implementation follows.

Structurally, the script consists of three building blocks which are the instruction, the segment, and the process. An instruction provides the means for defining player actions, assignments, or conditional logic/decision making requirements. The script segment is a continuous group of one or more script instructions beginning with a labeled instruction and terminating with a "wait" instruction. A process contains one or more script segments which are related to a common scenario threat or theme. In a typical engagement scenario involving two opposing forces there would be at least two script processes (a minimum of one script process for each side). The script structure is illustrated in figure 1.

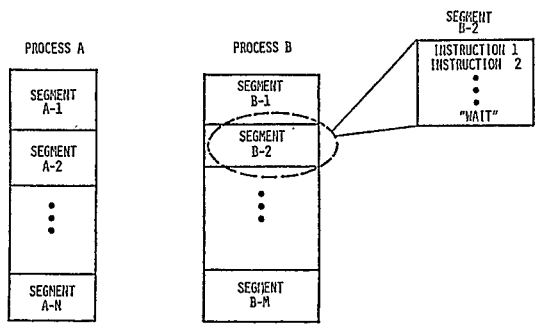


Figure 1. Script Structure

A functional overview of the script processing methodology is presented in figure 2. This figure is referenced in the discussion of the script processing methodology that follows.

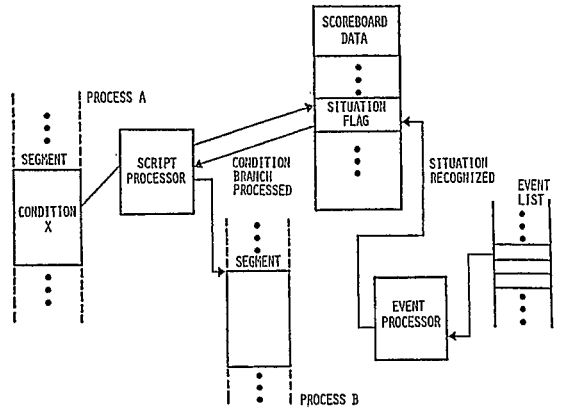


Figure 2.
Functional Overview of the Script Processing Method

In script processing, there are two model execution states. The first state is embodied in the processing of the script instructions by the script processor. The second state consists of the processing of script-generated events by the event processor.

The processing of script instructions is the initial model execution state. The user selects via the script input language which script processes are to be processed first. The processing of script instructions is performed sequentially starting with the first instructions of the first segment of each process. Action type instructions generate events. Conditional logic instructions generate "situation" flags that are stored in data structures called scoreboards. Scoreboards contain data which reflect the entire state and condition of execution of the computer model. Each process has a scoreboard that defines associated players, active contingencies (that is, what can cause changes in the activities of the players) and the location in the script that is currently being processed (if the process is not in the "wait" state defined below).

The processing of script instructions continues until all processes are in the "wait" state (that is, WAIT instructions have been encountered in all processes). When this occurs, model execution shifts to the event processing state. Events that were generated in the script processing state are now executed from a time-ordered queue. Some events, such as player movement, may generate additional events referred to as micro events. In addition, the performance of certain events may cause future events to be cancelled. For example, when a player is killed, all of his future movement and firing events are deleted. In the case of movement,

these micro events are responsible for moving a player from one point to another by taking into account terrain features and opposing player positions. The event processing state continues unless a "situation" is recognized (satisfied). If this occurs, model execution shifts to the processing of script instructions that are specified by the conditional logic instruction (that defined the "situation"). The script processing state will continue until the "wait" state is reached and then, model execution will again shift to the event processing state. Model execution will continue in this fashion until all events and script instructions have been processed.

THE TRANSPORTATION SAFEGUARDS EFFECTIVENESS MODEL

A model that effectively utilizes the script processing technique is the TSEM (Transportation Safeguards Effectiveness Model). TSEM is a monte carlo, event-stepped computer model which plays a two-sided engagement between courier defender units of an overland shipment of special materials and terrorist attacker units of varying degrees of sophistication and training. An overview of the TSEM functions and capabilities follows.

The players in TSEM (i.e., persons and vehicles) are directed by a script which includes actions (movement, firing, and dismounting) and contingency situations (player or vehicle at a location, players dead or incapacitated, attack started). The script is reentrant and several sections may operate in parallel to direct different groups of players. At least two parallel sections operate concurrently; one for the attackers, and one for the defenders. The script language has statements to create a new parallel section (called a script process) and associate it with a group of players. Associated with a script is a user defined data file containing information that establishes intitial conditions for the scenario (for example, location of attackers at the start of the scenario, initial number of rounds per clip, etc.). The implemented script instructions are summarized in table 1.

Table 1. TSEM Script Instructions

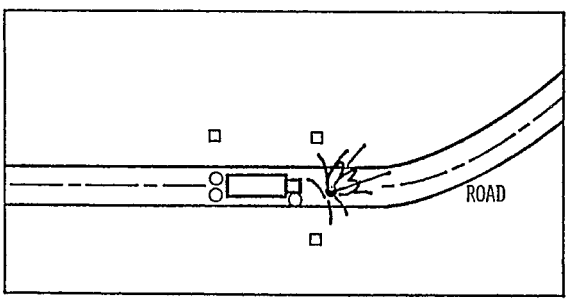
INSTRUCTION	COMMENTS
START LABEL 1, LABEL 2;	Activate script processes labeled LABEL 1 and LABEL 2.
WAIT;	Terminates a script segment.
PENETRATE;	An instruction to attacker forces to attempt to penetrate the cargo vehicle and remove the special materials.
FIRE, FIRER = PERSONLIST (WEAPON), TARGET = PLAYERLIST;	Instruction to fire a weapon with type defined by WEAPON.
WHEN (PLAYERLIST CONDITION) THEN GO TO LABEL;	Conditional logic/decision making command, implemented conditions are ACTIVE, INCAPACITATED, ATTACKED and AT (X, Y, Z).
GET (PLAYERLIST);	Associates player(s) or vehicle(s) with a script process.
MOVE, MOVEE = PLAYERLIST, GOAL = (X, Y, Z), SPEED = REAL NUMBER;	Moves player(s) or vehicle(s) from present location to GOAL.
DISMOUNT, DISMOUNTEE = PERSONLIST, GOAL = (X, Y, Z), SPEED = REAL NUMBER;	Similar to MOVE except specialized for deployment from a vehicle(s).

The battles take place on a two-dimensional surface with terrain and vegetation superimposed. The line-of-sight interruptions due to vehicles, terrain and vegetation features are correctly calculated and taken into account in firing allocation (i.e., people will not fire at opponents they cannot "see"). The battles progress until all people on one side are killed off, a preset time limit is reached, or cargo vehicle penetration is successfully completed. A movie can be produced which shows the course of battle, including player movements and shots fired.

Finally, TSEM features a rather detailed casualty assessment submodel (BDM Report, 1977). This TSEM submodel includes postures (standing, crouching, prone, kneeling, sitting), presented angle, and possible immediately adjacent shielding. The vulnerable areas of the body are approximated by rectangles and the appropriate conditional probability of incapacitation (given a hit) is used on each one. The TSEM casualty assessment submodel represents a major improvement over other existing models (for example SIAF, 1973 and ASARS, 1973) with respect to the level of detail utilized in modeling the human body in a variety of postures and shielding aspects; with respect to the use of conditional lethality data for various body parts; and with the respect to the fact that battle times were increased by approximately an order of magnitude based on comparisons with the SIAF (Small Independent Action Forces) model (SIAF, 1973).

EXAMPLE SCENARIO AND SCRIPT

In figure 3, a snapshot of an example TSEM scenario during execution is presented.



○ DEFENDERS (3), DU1

□ ATTACKERS (3), AU1

▭ CARGO VEHICLE (1), DV1

Figure 3. TSEM Scenario Snapshot

In this example, a vehicle carrying special materials has been stopped along a road due to the detonation of an explosive device placed by an attacker force of three members. Also depicted in this figure, are three members of the defender force deployed outside of the cargo vehicle in defense of that vehicle. For convenience, the three defenders have been grouped as defender unit 1 (DU1), the three attackers have been grouped as attacker unit 1 (AU1) and the cargo vehicle is referred to as defender vehicle 1 (DV1). The objectives of the attacker force are (1) to kill (incapacitate) all of the defenders, and (2) to penetrate the cargo vehicle. Conversely, the defender force goal is to kill all of the attackers. The portion of the script that describes the scenario snapshot is presented in figure 4. A brief discussion of each script instruction for this example follows.

```
        START ATT1, DEF1;
ATT1:   GET (AU1);
        FIRE, FIRER = AU1(10), TARGET = DU1;
        WHEN (DU1 INCAPACITATED) GO TO ATT2;
        WAIT;
ATT2:   PENETRATE;
        WAIT;
DEF1:   GET (DU1);
        DISMOUNT (DU1);
        FIRE, FIRER = DU1(8), TARGET = AU1;
        WAIT;
         •
         •
         •
```

Figure 4. Example Script

The START instruction informs the script processor of the two names of the processes to begin processing. The first process which defines the attacker scenario consists of the two segments labeled ATT1 and ATT2. As depicted in this example script, the defender force process consists of only one segment (labeled DEF1). The GET instruction in the segment ATT1 associates attacker unit 1 with this process. The FIRE command instructs attacker unit 1 to fire at defender unit 1 with weapon type 10. It is the event processing software that performs the required line-of-sight, detection and identification calculations and the associated casualty assessment determinations. If the condition specified in the WHEN script instruction is satisfied, then segment labeled ATT2 will then be processed. This segment contains only the PENETRATE instruction. The PENETRATE instruction is a special TSEM macro command. For this example, this macro command directs that all surviving attackers move to the cargo carrying vehicle to attempt to penetrate this vehicle. The GET instruction in the segment labeled DEF1 associates all members of defender unit 1 with the defender process. The DISMOUNT command is another example of a macro command. It directs that all members of defender unit 1 move from the cargo vehicle and deploy themselves at predetermined locations around the cargo vehicle. The FIRE command directs that defender unit 1 fire at the members of attacker unit 1 with weapon type 8.

TSEM GRAPHICS

Even simple scripts of force-on-force engagements generate scenarios that are complex to analyze when the simulation output involves only a printer listing. The positions, movements, and interactions of the players amongst themselves and with their environment (scene) are difficult to visualize at best. One can spend hours analyzing why player defender 1 didn't fire upon player attacker 3 at 5 minutes into the fire fight. Was he out of ammunition? No, we check the scoreboard of player data and see that he has several clips left. Was he wounded and incapacitated so that he couldn't fire? No, the data reveals that he was not wounded. Was he moving to a new location or involved in some other activity which occupied his full efforts so that he didn't have time to fire? No, he was just standing in a fixed area. Did he have line-of-sight to attacker 3, and so forth?

This and other necessary analysis efforts dictate computer graphics for information display and, in particular, animated motion pictures. Even the most sophisticated and elegant simulation is worthless if the results of the simulation cannot be interpreted. Large engagements involving many players and complex interactions producing reams of listings may lend themselves to misinterpretations and negative effects. This is especially true where human lives are potentially involved, as in wargame simulations resulting in policy decisions. Thus, the graphical illustration of simulation results is a necessary and integral part of any such code. Indeed, they are inseparable. With this basis, then, the mechanics of constructing the complete simulation with script, scene, action and motion picture results are discussed in the following paragraphs.

An intriguing situation which arises in computer simulations, such as TSEM, utilizing graphics for information display, is in the definition of the player universe. Initial development possibilities fall into three categories. First, the scene can be defined and the script written to utilize its peculiarities. Secondly, a given script might indicate the best way to construct the terrain. Finally, the scene and script can be developed simultaneously to maximize the interactions of player and environment. In some instances, the choice is dictated by that element which is of greater importance.

For example, if the simulation is designed to evaluate defense configurations against penetration at a nuclear fuels repository, then the scene or layout is fixed. The attackers are varied in numbers and configurations, as are the defenders, along with weapon types, motivation, a priori knowledge and so forth. The script which directs the player interactions and governs decisions during the course of the engagement is also varied. Here, and especially in the case of an established facility, the terrain is known and fixed, and the script controls all the what ifs' in repetitive runs of the simulation. The opposite situation occurs when the scene is constantly changing and the script remains constant. In this situation, a particular defender and attacker configuration and script are analyzed in varying environments to test for wide applicability. In the example, a fixed defender force can be implemented at different sites and then the engagements simulated. Finally, the most interesting case is that where both the script and the scene can be varied. This is usually not done in any one particular computer run because of the complexities involved in the analysis and comparison of the results. Thus, there exists an interactive bond between the script writer, the scene layout producer and the analyst in the classic war gaming simulation environment.

Separating the elements of scene and script, momentarily, it will be assumed that some agreement or plan has been reached in a specific force-on-force engagement. It is now necessary to generate a description of the player universe in a form that can be processed by a high speed digital computer and that is compatible with the rest of the simulation code. For TSEM several factors dictated a macro-terrain structure generated by x, y - coordinate data and accompanying elevation information in the z - axis. The most important of these factors was the desire to reproduce a known terrain via a U.S. Geological Survey Data tape formatted in the Universal Transverse Mercadian (UTM) mode. These tapes were produced by aircraft or satellite mapping and have elevation resolution accurate to 1 meter. Programs exist which can strip out any particular window in degrees, minutes and seconds of latitude and longitude, producing a file of the area of interest. This file may also be produced via the brute force method of digitizing a contour map of the desired area, or an interactive graphics package and CRT can be utilized to generate the data file. These last two methods are not as accurate or as fast in replicating the existing terrain as the UTM data tape. A second factor for the mathematical description of the terrain is in the line-of-sight algorithms in the fire allocation subroutines. These algorithms require coordinate position information along with elevation data to perform their calculations. Any terrain information not in this form would have to be converted and duplication of information would result. Figure 5 illustrates the various input methods discussed. The final result is a file of coordinate and elevation data which is utilized as a "ground-work" for the addition of micro-terrain features.

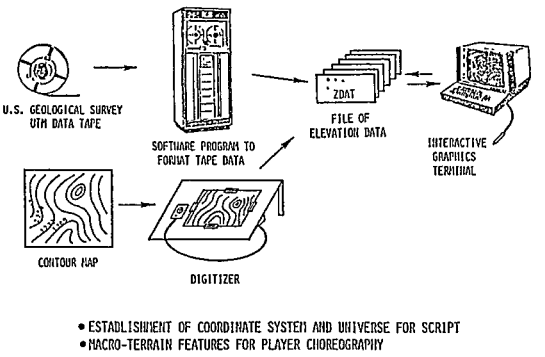


Figure 5. Generation of Script Action Environment

This file can be displayed on a CRT for viewing and further analysis can be done such as generation of lines of constant elevation to produce a contour map. Figure 6 shows a 3-dimensional display of some macro-terrain data and the projected contour lines. The generated contour map along with the 3-dimensional image is then used to check the macro-terrain data file for accuracy and any changes accommodated either interactively with the CRT or by appropriately editing the data file in a batch mode.

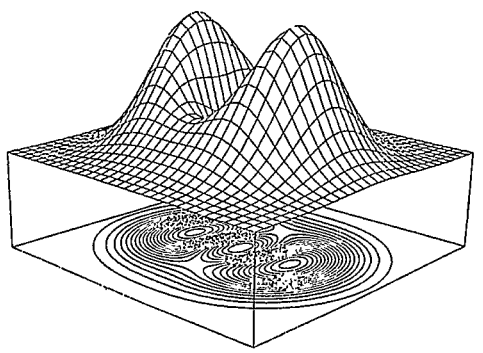


Figure 6. Macro-Terrain Data

Building upon the macro-terrain we can add micro-terrain features to customize the landscape and simulate various seasons and weather conditions. The process was implemented in TSEM by using polygons placed within the coordinate system. Within any particular polygon a vegetation class, soil type, or positive or negative undulations were defined. Thus, vegetation classes such as heavy forest, swampland, savannah brush or high mesa can be implemented; soil types such as sandy or hard packed clay defined (which influences mobility); and individual

boulders placed. Also, obstacles such as ravines, ditches, lakes and rivers and their dimensions can be defined and located. Figure 7 shows an example of micro-terrain features. Again, as with the macro-terrain, this file can be edited to produce the desired scene layout.

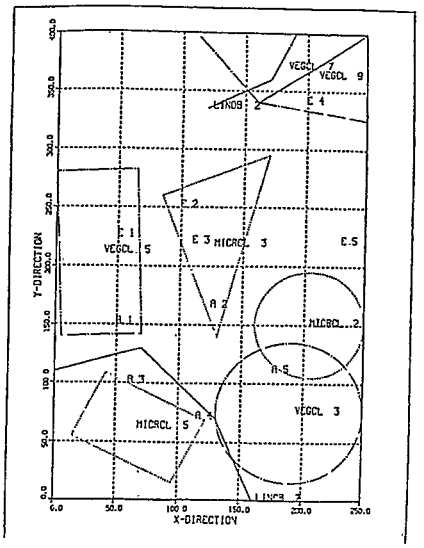


Figure 7. Micro-Terrain Vegetation Polygons

Finally, the macro and micro-terrain features combine to completely define the player universe as shown in figure 8.

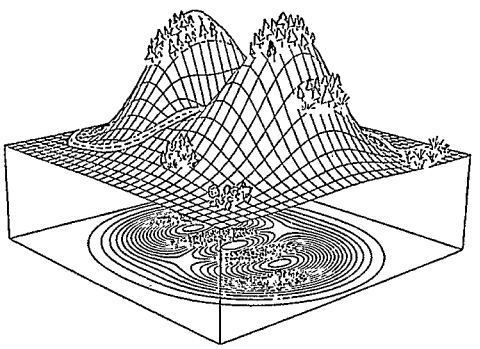


Figure 8. Terrain Data

The established environment of the fire-fight can serve to bound the endless possibilities of valid scripts. One would not knowingly position attackers where they would not have line-of-sight to their targets, nor would one direct players to move to a goal that they cannot possibly reach (viz. across an unnegotiable ravine). Thus, the script can more realistically represent viable realities and options in the physical world. Hence, the off-line development of the terrain data file serves to shape the script in addition to being directly involved in the simulation calculations and utilized in the graphics program, as shown in figure 9.

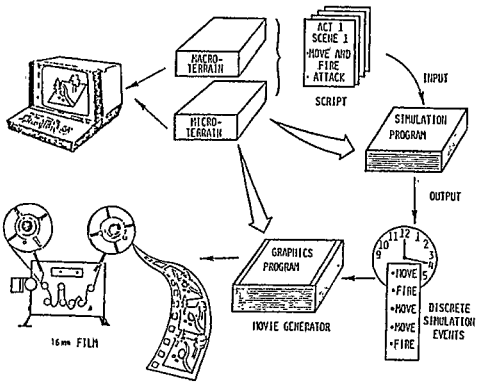


Figure 9.
The Script Driven Simulation and Graphics

As the TSEM execution progresses, the simulation events are produced by processing the script. These events are formatted and collected in a file to later drive the movie generating program. An example of a fire event and a movement event are shown in figures 10 and 11, respectively.

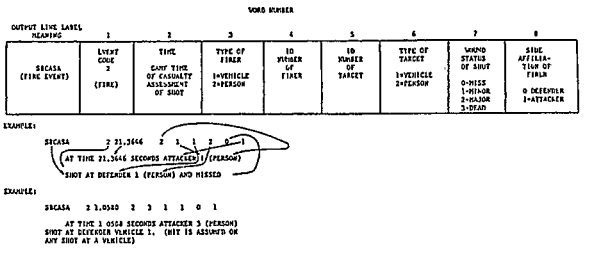


Figure 10. Movie Fire Event Output

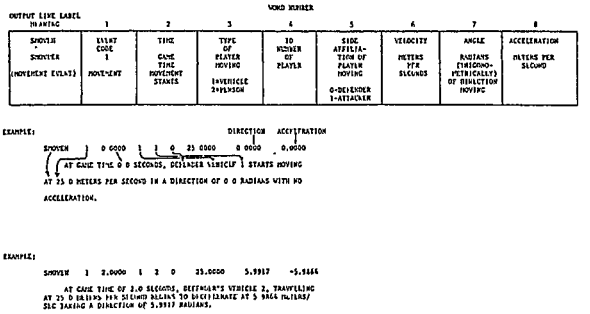


Figure 11. Movie Movement Event Output

It is important to note that different types of event vectors are handled differently within the graphics code. A fire event produces a vector drawn on the screen from the firer to its target at the time of the shot. This is then repeated for 1 second worth of viewing

time on motion picture film for analysis purposes. A movement event is much more involved in its graphical representation. This is due in part to the translation from discrete events to the analogue film medium. The player's position must be updated each frame based on its last position, direction, speed, and acceleration or deceleration, if any. This requires memory from frame to frame in addition to the calculations updating the data every 1/24th of a second.

The illustration in figure 12 represents a frame from a TSEM computer-generated movie of a firefight between attacker forces deployed at the ambush sight, and defender forces traveling through the site in various convoy vehicles. Referring

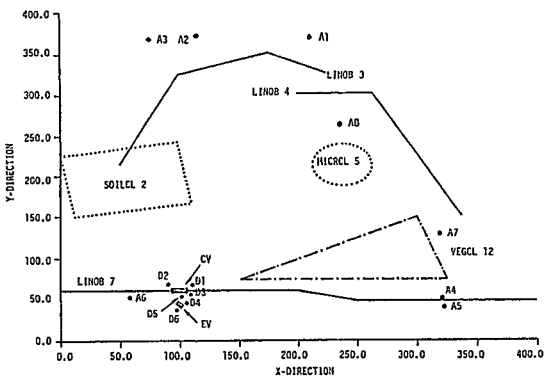


Figure 12. Terrain-Vegetation Scenario Example

to figure 12, terrain and vegetation features include a heavy forest (VEGCL 12), sandy soil (SOILCL 2), boulders (MICRCL 5), a ravine (LINOB 3), and a dike (LINOB 4) situated as shown near the ambush site. The defender cargo vehicle (CV) and escort vehicle (EV) have been stopped along the road (LINOB 7) and defender players 1 through 6 (DI-D6) have dismounted. After stopping the convoy by detonating explosives in the road or firing upon the vehicles, the attackers (A1-A6) then proceed to engage the defender unit. (In a TSEM movie, a general overhead view (or plan view) is maintained and micro-terrain features eliminated to avoid clutter and to view the entire firefight. A zoom close-up is incorporated toward the end of the film as the attackers attempt to penetrate the convoy security. Additional information such as casualty lists for both sides and scenario clock are included as the battle progresses.)

SUMMARY AND CONCLUSIONS

In summary, a detailed description of the script processing technique and the innovative implementation of this technique in the TSEM model and the application of this approach to the generation of animated computer graphics have been presented. Based on user comments, the script processing technique as implemented in the TSEM provides a very powerful user interface medium in terms of ease of use and flexibility in defining complex scenarios. The use of the TSEM Analyses of TSEM outputs, especially the movies, has provided users with valuable information pertaining to the evaluation of attacker/defender tactics, the duration of the simulated battles, and response force requirements.

ACKNOWLEDGEMENT

The author wishes to acknowledge the collection of subroutines found in the terrain-vegetation submodel modified from the SIAF code of TRW by Pat DeLaquil of Sandia National Laboratories and John Dawson (former employee of The BDM Corporation, Albuquerque). He would also like to acknowledge Dr. Bruce D. Link (formerly of Sandia National Laboratories) for his technical contributions under contract 05-9365 and Mr. Dawson for his development of the graphics package. In addition, the author acknowledges the significant contributions of the other BDM Corporation project technical staff including Messrs. J. L. Cooke, S. D. Jones, and H. G. Pringle, and Mr. L. H. Skinner (former BDM employee).

REFERENCES

The Probability of Single Shot Incapacitation, P(SSIN), Algorithm for TSEM, Report no. BDM/A-77-274-TR, The BDM Corporation, Albuquerque, New Mexico, June, 1977.

SIAF System Model User's Manual: Small Independent Action Forces, TRW Systems Group, Report no. 20660-6007-R0-00, Volume VI, "Combat Execution Subroutines," December 1973.

SIAF Model Development Validation and Implementation, Final Report, TRW Systems Group, August 1971. Volume I, Report no. 16905-6012-R0-00
Volume III, "Model Subroutines (Terrain, Weather, Targets)," Report no. 16905-6014-R0-00

ASARS Battle Model, United States Army Combat Development Command, Systems Analysis Group, Report no. USACDCSAG-TR-9-73, March 1973.
"Book 2, Volume II-A, Narrative Description"
"Book 5, Volume II-B, Phase B and C Charts"
"Book 9, Part 1, Volume III, User Manual"

AUTHOR'S BIOGRAPHY

ROBERT E. SHERIDAN, JR. was born on January 18, 1943 in Wilson, PA. He received the B.S. and M.S. degrees in Aerospace Engineering from The Pennsylvania State University in 1965 and 1968, respectively. In 1973, he received the M.S. in Computer Science from The Pennsylvania State University.

Mr. Sheridan was employed by the Boeing Company from 1968 to 1970, Pratt & Whitney Aircraft-Florida Research and Development Center from 1970 to 1971, the Applied Research Laboratory as a graduate research assistant from 1971 to 1973 in the areas of theoretical and experimental fluid mechanics. Since 1974, he has been employed by The BDM Corporation and is currently a Principal Staff Member. His work has been primarily concerned with the development of computer models and simulations. He has been assigned technical leadership and management positions on numerous projects.

Mr. Sheridan is a member of the ACM, IEEE Computer Society, and the AIAA.

The BDM Corporation
1801 Randolph Rd. SE
Albuquerque, NM 87106
(505) 848-5000