

MARTIN MARIETTA ENERGY SYSTEMS LIBRARIES



3 4456 0367323 9

# oml

ORNL/TM-11973

1700  
5

**OAK RIDGE  
NATIONAL  
LABORATORY**

**MARTIN MARIETTA**

## Graphics Development of DCOR: Deterministic Combat Model of Oak Ridge

Greg Hunt  
Y. Y. Azmy

OAK RIDGE NATIONAL LABORATORY  
CENTRAL RESEARCH LIBRARY  
CIRCULATION SECTION  
4500N ROOM 175

**LIBRARY LOAN COPY**

DO NOT TRANSFER TO ANOTHER PERSON

If you wish someone else to see this  
report, send in name with report and  
the library will arrange a loan.

ORNL-158 (3-87)

MANAGED BY  
MARTIN MARIETTA ENERGY SYSTEMS, INC.  
FOR THE UNITED STATES  
DEPARTMENT OF ENERGY

This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from the Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, TN 37831; price available from (615) 576-8401, FTS 626-8401.

Available to the public from the National Technical Information Service, U.S. Department of Commerce, 5235 Port Royal Rd., Springfield, VA 22161.

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ORNL/TM-11973

100  
5

**Graphics Development of DCOR:  
Deterministic Combat Model of Oak Ridge\***

Greg Hunt  
Georgia Institute of Technology

Y. Y. Azmy  
Engineering Physics and Mathematics Division  
Oak Ridge National Laboratory

DATE PUBLISHED - OCTOBER 1992

Prepared by the  
Oak Ridge National Laboratory  
Oak Ridge, Tennessee 37831  
managed by  
Martin Marietta Energy Systems, Inc.  
for the  
U.S. DEPARTMENT OF ENERGY  
under contract No. DE-AC05-84OR21400



3 4456 0367123 9



## TABLE OF CONTENT

	Page
ACKNOWLEDGMENT .....	v
ABSTRACT .....	vii
I. INTRODUCTION .....	1
II. CONVERSION OF DCOR'S DISCRETE-VARIABLE SOLUTION TO A PIXEL MAP .	3
III. IMPLEMENTATION INTO DCOR CODE .....	7
IV. DESCRIPTION OF DCOR'S GRAPHICS SUBROUTINES .....	8
V. EXPLANATION OF NEW INPUT ENTRIES .....	13
VI. CONCLUSION .....	15
REFERENCES .....	16
APPENDIX A - Obtaining NCSA Software (XImage and HDF Library) .....	A-1
APPENDIX B - Format of the ASCII Bitmap Files B-1 .....	B-1
APPENDIX C - Format of the ASCII Background File .....	B-1
APPENDIX D - Performance Data .....	D-1
APPENDIX E - Example Input Files for Demonstration Simulation .....	E-1
APPENDIX F - Example Frames from Demonstration Simulation Run #1 .....	F-1



## ACKNOWLEDGMENT

We would like to thank the Visualization Lab at Oak Ridge National Laboratory, in particular Brian Wallace and Ross Toedte, for their help in producing the illustrations and accompanying video tape and in obtaining the public domain software from NCSA. Also, we would like to thank V. Protopopescu and R. T. Santoro for many enlightening discussions during the course of this work



## ABSTRACT

DCOR is a user-friendly computer implementation of a deterministic combat model developed at ORNL. To make the interpretation of the results more intuitive, a conversion of the numerical solution to a graphic animation sequence of battle evolution is desirable. DCOR uses a coarse computational spatial mesh superimposed on the battlefield. This research is aimed at developing robust methods for computing the position of the combative units over the continuum (and also *pixeled*) battlefield, from DCOR's discrete-variable solution representing the density of each force type evaluated at gridpoints. Three main problems have been identified and solutions have been devised and implemented in a new visualization module of DCOR. First, there is the problem of distributing the total number of objects, each representing a combative unit of each force type, among the gridpoints at each time level of the animation. This problem is solved by distributing, for each force type, the total number of combative units, one by one, to the gridpoint with the largest calculated number of units. Second, there is the problem of distributing the number of units assigned to each computational gridpoint over the battlefield area attributed to that point. This problem is solved by distributing the units within that area by taking into account the influence of surrounding gridpoints using linear interpolation. Finally, time interpolated solutions must be generated to produce a sufficient number of frames to create a smooth animation sequence. Currently, enough frames may be generated either by direct computation via the PDE solver or by using linear programming techniques to linearly interpolate intermediate frames between calculated frames.



## I. INTRODUCTION

As an alternative to computationally expensive stochastic combat models, ORNL developed a deterministic combat model based on a set of Partial Differential Equations (PDEs).<sup>1-5</sup> This new model differs from earlier Lanchester-type deterministic models that are based on Ordinary Differential Equations (ODEs) in that it explicitly includes the spatial aspect of battle.<sup>6-8</sup>

The Deterministic Combat model of Oak Ridge (DCOR) code is an interactive, menu-driven computer implementation of this model.<sup>9</sup> As DCOR is based on a deterministic model, a finite spatial mesh is superimposed on the battlefield, and DCOR's solutions are restricted to the gridpoints of this mesh. Also, as the solution to the mathematical model is composed of large numerical arrays, to diversify the "useability" of DCOR, a conversion from this numerical solution to a graphical animation sequence is desirable. Therefore, a process to convert the numerical solution of DCOR, the density of each force type at computational mesh points (CMPs) and the total number of units in each force, into a graphical solution has been developed, implemented, and tested. In this TM we use the terms *force type* or *weapon system* to denote a troop/weapon combination characterized by a specific attrition capability, resistance to enemy fire, mobility, maneuverability,...; e.g., infantry, tanks, smart weapons, etc. On the other hand we use the terms *combative unit*, or *object* to denote a military entity formed of a single force type; on the fine-most scale each object represents an individual soldier, tank, etc, but on coarser scale, may be used to represent a platoon, company, battalion, and so on. Also, by *battlefield* we mean an area of land, including terrain features such as shores, mountains, forests, etc, sufficiently large to contain the entire engagement and aspects of its aftermath, such as retreat and pursuit, that is desirable to model.

The animation sequence can be generated either in a series of ASCII files or in a single file in the National Center for Supercomputing Applications's (NCSA's) Hierarchical Data Format (HDF).<sup>10</sup> The ASCII files are easily modified to allow importing of the animation sequence into almost any graphics package with importing and animation capabilities. The HDF file may be viewed using one of NCSA's public domain software packages such as XImage, XDataSlice, or DataScope. These packages and the HDF library with the functions used in DCOR's graphics subroutines are available from NCSA (see Appendix A).

This document describes the development of this visualization process and its implementation into a new visualization module of the computer code DCOR. In Section II, we develop the theoretical problems and solutions in converting DCOR's numerical solution to an animation sequence. Section III is concerned with the implementation of the conversion process developed in Section II into the code. Section IV describes in detail the graphics subroutines that have been added to DCOR. Next, we provide descriptions and explanations of the new graphics parameters in Section V. Then, we summarize our efforts and discuss future

developments for the visualization module of DCOR in Section VI. Appendix A provides instructions for obtaining public domain software from NCSA. Appendices B and C describe the format of the output ASCII files and the format of the background bitmap ASCII file respectively. Finally, Appendix D provides some comparison tables for the two demonstration simulations produced by DCOR so far.

## II. CONVERSION OF DCOR'S DISCRETE-VARIABLE SOLUTION TO A PIXEL MAP

Since DCOR is based on a deterministic PDE model, the problem domain, ie. the battlefield area, is divided into finite, local subdomains using a finite spatial mesh. All input parameters and the computed solution are restricted to their corresponding values evaluated at the gridpoints of this mesh.<sup>9</sup> DCOR then uses a modified version of the public domain PDE solver, PDE TWO to solve for the time dependent density of each of the modeled force types at the computational mesh gridpoints. First, PDE TWO converts the set of PDEs into a set of ODEs via the Method of Lines<sup>11</sup> by approximating the spatial derivatives on the computational mesh using a finite-difference scheme. PDE TWO then uses the GEARB<sup>12</sup> method to solve the resulting set of ODEs. Thus, the output of the PDE solver is the density of each weapon system or force type,  $m$ , at each CMP,  $(i,j)$ , at each time step,  $t$ ; we denote this density by  $u_{mij}(t)$ . Previously, this solution was presented to the user as a static sequence of contour plots. However, as an animation sequence depicting the movement and interaction of the individual units of the different weapon systems is easier to interpret, a conversion of the discrete-variable solution to such animation sequence is highly desirable. An animation sequence consists of consecutive frames, each representing a bitmap, shown in chronological order that must be displayed in rapid succession, in order to produce a realistic visual effect of temporal evolution.

The graphical display of most of today's computers is composed of an  $a \times b$  grid of small dots called pixels. A bitmap is an  $a \times b$  integer array where each integer in the array represents the corresponding pixel and the value of the integer determines which color from an array of colors, called a palette, is displayed at that pixel. Thus if we can represent DCOR's numerical output at each time step in the form of a bitmap, then we can display this bitmap on a computer screen as a single static frame. By displaying consecutive bitmaps in rapid succession we obtain the desired animation sequence. Since a single pixel is extremely small, and difficult to see, it is necessary to represent each object as a  $n \times n$  square of pixels. We define an  $a/n \times b/n$  transitional mesh in which each transitional mesh point (TMP) represents a corresponding  $n \times n$  square of pixels on the bitmap, so that an object can be displayed only at a TMP. Since we are able to distinguish different forces by using different integer values (corresponding to different colors) in the bitmap, the process of generating a bitmap from the numerical solution for one force can be repeated for each of the remaining forces and the resulting bitmaps superimposed to produce a single frame. Hence, the objective of converting a numerical solution on a computational mesh into a graphical solution can be viewed as converting a real-valued numerical solution on a computational, possibly nonuniform, mesh to an integer numerical solution on a uniform transitional mesh. This objective can then be divided into three main problems: (i) distributing the total number of units among the CMPs; (ii) distributing the total number of units at a CMP over the area

attributed to that point; and (iii) producing a sufficient number of frames of properly interpolated output to create a smooth animation sequence.

(i) The PDE solver, produces the density of each combat force type, i.e. number of combative units per unit area of the battlefield, at each CMP, which when integrated over the entire area of the battlefield produces the total number of units for this force type. Since fractions of units are not acceptable in the visual interpretation of the results, we round each force type's total number of units to the nearest integer. In addition, from the x- and y-mesh inputs, we can calculate the computational cell area attributed to each CMP. Then, by multiplying the local density by the area, we obtain a real number value, that we denote  $unum_{ij}$ , for the number of units allocated to each CMP. If we simply rounded each real number to the nearest integer, the sum of the integers may or may not equal the total number of units. Thus we examine each  $unum_{ij}$ . If  $unum_{ij}$  is equal to or greater than one, we allocate a number of units equal to the integer portion of  $unum_{ij}$  to the (i,j)-th CMP. We then reduce both  $unum_{ij}$  and the total number of remaining units to be allocated by this same amount. This process will distribute a number of units less than or equal to the total number of units to the CMPs, and at its conclusion  $unum_{ij}$  will be less than one for all  $i$  and  $j$ . At this point we assign any remaining units to the CMPs with the largest  $unum_{ij}$  fractional values.

After examining the output generated by this process, we found that in certain instances, particularly when a force is split to maneuver around natural or man-made obstacles on the battlefield, a unit may appear to fortuitously oscillate between distant points on the mesh. This occurs because density changes at distant points produce small fractional differences in  $unum_{ij}$ , making it larger at one point for a few time steps, then smaller for a few time steps, etc. Therefore, we employ a linear programming algorithm originally developed for the transportation problem to suppress this visually distracting behavior. The transportation problem involves routing units from several sources to several destinations in order to minimize the total cost (distance) between the sources and the destinations.<sup>13</sup> In solving the transportation problem, there is a method, called the Big M<sup>13</sup> method, of assigning an exceptionally high distance, hence cost, between a source and destination which make an undesirable or impossible match. Thus, to employ this algorithm, we use the CMPs and their allocated units from the previous frame as the sources, and for the current frame we use the process described above to distribute units to the CMPs, which then represent possible destinations. In choosing the possible destinations, a few more than the total number of units are chosen to allow some degrees of freedom for the algorithm, otherwise the solution would remain unchanged. In addition, they are separated into two categories:

1. integer destinations: destinations requesting a number of units equal to the integer portion of  $unum_{ij}$ ;
2. fractional destinations: destinations corresponding to a  $unum_{ij}$  value of less than one.

As a requirement of the problem, the number of units provided by the sources must equal the number of units requested by the destinations. If these are not equal, a dummy source/destination must be introduced providing/requesting a number of units equal to the deficit/surplus.<sup>13</sup> In determining the output, destinations receiving units from a dummy source are not used. In order to represent the true solution as much as possible, it is desired that integer destinations receive units before fractional destinations. If a dummy source is needed, a match between a dummy source and an integer destination is undesirable, therefore, we use an exceptionally large value for the distance between integer destinations and the dummy source. Otherwise, a value of zero is used for the distance between the dummy source and fractional destinations. Also, we introduce an input for maximum distance a unit is allowed to travel between frames. If the distance between a source and possible destination is greater than the maximum distance allowed, an exceptionally large value is used for the distance instead.

(ii) By design of the transitional mesh, each unit to be displayed must lie on a TMP, so that each unit assigned to a given CMP must occupy a TMP located within the area attributed to this CMP. Thus, the second problem constitutes the distribution of the units allocated to each CMP in stage (i) over the TMPs in its neighborhood. We accomplish this by interpolating the density function of each TMP from the computed densities of the CMPs surrounding it, via an equation of the form:

$$u(x,y) = Ax + By + Cxy + D$$

where  $x$  and  $y$  are the coordinates of the TMP with respect to a locally defined coordinate system;  $A$ ,  $B$ ,  $C$ , and  $D$  are coefficients calculated using the value of  $u$  at the CMPs surrounding this TMP at the given time step. After calculating the density at each TMP within the area of the CMP, we then distribute the units, already assigned to this CMP, one by one, to the TMP with the largest density. Since each TMP can contain only one unit, we set to zero the density at any TMP to which a unit has been assigned, to preclude multi-unit assignment to the same TMP upon subsequent comparison of TMP densities. If there are not enough TMPs within the CMP to hold all the units, the user is notified and it is suggested that the user increase the scale factor relating the computational mesh and the transitional mesh (make the transitional mesh finer).

(iii) Upon monitoring the performance of the PDE solver on a few test problems, we found that the determining factor of the amount of CPU time consumed in solving a problem is the final time level, or time length of the simulated battle and not the number of time steps at which the solution is generated. Hence, generating the numerical solution for ten time steps is just as costly as generating the numerical solution for one hundred time steps for the same final time level. If the PDE solver is later upgraded, this may not be the case. Therefore, a method of producing intermediate frames between two calculated frames has been developed.

If we can identify a unit's position in two calculated frames, then we can generate intermediate frames by using linear interpolation between its initial and final positions in order to determine the positions of this unit in all intermediate frames. To accomplish this task, we once again consider linear programming techniques. In particular, we look at the assignment problem<sup>14</sup>, which is a special case of the transportation problem. In this variant, the number of sources must equal the number of destinations, and each source can provide only one unit and each destination can accept only one unit. In this case, the sources are the positions of the units on the transitional mesh in the earlier calculated frame and the destinations are the positions of the units on the transitional mesh in the currently calculated frame. The algorithm then matches the units in the two frames by minimizing the total cost (distance); clearly in the present application we are not concerned with this minimization, but the algorithm serves our purpose. Also, the total number of units provided by the sources (in this case the total number of sources), and the total number of units requested by the destinations (in this case the total number of destinations), must be equal. However, since a source/destination can only provide/request one unit, then additional dummy sources/destinations may have to be added in to balance the problem.<sup>13</sup> In addition only the units which have a match in both calculated frames will appear in the interpolated frames. Hence, units will not be attrited, nor will new units be added in any intermediate frames. Once the algorithm concludes the matching process the desired (user-specified) number of intermediate frames are generated using linear interpolation between the two calculated frames. To prevent two units from occupying the same TMP, if the calculated position of a unit is already occupied, then another position along the line segment connecting the unit's positions in the two frames is chosen. Currently, there is no discernable CPU time difference between interpolating frames and calculating frames. However, in larger simulations a difference may become apparent. The visual advantage of interpolation is the smoother animation it is capable of producing due to a more evenly spaced movement rate between calculated frames. The precision advantages of direct computation of frames include the timely display of attrition and a more accurate depiction of maneuvering. The flexible implementation of the visualization module into DCOR allows the user to compromise these conflicting advantages according to his own taste and specific application.

### III. IMPLEMENTATION INTO DCOR CODE

The visualization graphics subroutines for DCOR have been developed and are currently operating on ORNL's Cray X/MP under the UNICOS operating system.<sup>9</sup> In order to preserve the portability of the DCOR code, the additional graphics subroutines have been written in standard FORTRAN. However, the subroutine **grafix** uses the linear programming subroutine, **H03ABF**, from the Nag library to solve the transportation problem. This should be easily modified to any other numerical software with a similar subroutine. Also, to generate the animation sequence in HDF format, the subroutine **raster** uses functions from NCSA's public domain HDF library, *libdf.a*. The C and FORTRAN source files needed to compile this library are available from NCSA through anonymous ftp or regular mail (see Appendix A).

The graphics subroutines are called after each time step if either the ASCII or HDF format is selected. They currently use a separate set of integer-, real-, and character-variable container arrays from the PDE solver subroutines. These arrays are setup and managed by **gsetup**. If these arrays are too small to perform the necessary calculations, **gsetup** notifies the user of the necessary sizes and terminates execution.

The methods described in Section II have been implemented into the graphics subroutines of the DCOR code. In particular, the initial distribution of the total number of combative units among the CMPs, where the transportation algorithm is not necessary, is performed in the subroutine **initgrafix**. For the remaining time steps, this allocation is performed within **grafix**. Next, the unit allocation process for each non-zero CMP to the associated TMPs is accomplished in **distribute**. Finally, the generation of any intermediate frames through linear interpolation is handled in the subroutine **interpol**.

In addition, a graphics modification menu has been added to the interactive menu so that the parameters described in Section V may be modified interactively.

#### IV. DESCRIPTION OF DCOR'S GRAPHICS SUBROUTINES

This section explains the new graphics subroutines for DCOR. Each entry contains the subroutine name in bold face, followed by its arguments in italics and their variable type. This is followed by a brief description of the subroutine's function.

**subroutine assct** (*N, A, C, T, maxn, CH, LC, LR, RH, SLC, SLR, U*)  
 INTEGER, *N, A(maxn, maxn+1), C(maxn), CH(maxn), LC(maxn),*  
*LR(maxn), RH(maxn+1), SLC(maxn), SLR(maxn), U(maxn+1)*  
 REAL *T*

This is a modified version of **assct**, a subroutine to solve the square assignment problem. The subroutine was obtained from the public domain NETLIB collection of subroutines. It has been modified to run using a container array.

**subroutine distribute** (*u, x, y, mx, my, mfrc, mxf, myf, maxn, dx,*  
*dy, MA, source, mi, isrce, nonzero, uf, nimage*)  
 INTEGER *mx, my, mfrc, mxf, myf, maxn, MA(mfrc),*  
*source(mfrc, 3, mx\*my), mi(mfrc), isrce(mfrc, 2, maxn),*  
*nonzero(2, mxf\*myf), nimage(mxf, myf)*  
 REAL *u(mfrc, mx, my), x(mx), y(my), uf(1, mxf, myf), dx, dy*

This subroutine distributes combat units over the transitional mesh points associated with each CMP. The allocation is performed by calculating the density at each TMP contained within the CMP and then distributing the total number of troops at the CMP individually to the TMPs with the highest densities.

Note: Each TMP can only contain 1 unit per weapon type.

**subroutine calc** (*uf, mxf, myf, dx, dy, x0, y0, A, B, C, D, i1,*  
*i2, j1, j2*)  
 INTEGER *mxf, myf, i1, i2, j1, j2*  
 REAL *uf(1, mxf, myf), dx, dy, x0, y0, A, B, C, D*

This subroutine calculates the interpolation equations for **distribute**.

**subroutine findmax** (*array, m, mfrc, mx, my, nonzero, nz, maxx,*  
*maxy*)  
 INTEGER *m, mfrc, mx, my, nonzero(2, mx\*my), nz, maxx, maxy*  
 REAL *array(mfrc, mx, my)*

This subroutine returns the coordinates of the largest value in an array where the coordinates of the non-zero elements of the array are stored in another array.

```

subroutine gbal (n, u, x, y, mx, my, mfrc, maxn, utot, unum)
INTEGER      n, mx, my, mfrc, maxn
REAL        u(mfrc,mx,my), x(mx), y(my), utot(mfrc),
            unum(mfrc,mx,my)

```

This subroutine calculates the balance table at every time step. It is essentially the same as subroutine `bal` except that the balance at each CMP is stored for graphics purposes.

```

subroutine grafix (x, y, mx, my, mfrc, dmax, MM, MB, MMM, utot,
                 unum, MA, source, nonzero, dest, KOST, K15, K7, K9, K6,
                 K8, K11, K12)
INTEGER      mx, my, mfrc, MM, MB, MMM, MA(mfrc),
            source(mfrc,3,mx*my), nonzero(2,mx*my), dest(4,MB),
            KOST(MMM,MB), K15(MM), K7(MM), K9(MM), K6(MM),
            K8(MM), K11(MM), K12(MM)
REAL        x(mx), y(my), dmax, utot(mfrc), unum(mfrc,mx,my)

```

This subroutine calculates the computational mesh graphics data for the initial frame. It first sets up an array of the possible locations, then evaluates a cost matrix whose entries are equated to the distance between each source and each possible destination. Then a dummy source/destination is added in if necessary. The transportation algorithm `H03ABF` from the NAG library is used to determine the desirable locations from the possible ones. Finally, any duplicate destinations in the list are combined and the destinations become the new sources.

Notes: (a) The possible locations are chosen based on the number of units (*unum*). Also, a value of 1 represents a whole unit. Desirable locations are chosen first based on the presence of whole units, and then from the locations with the largest fractional number of units.

(b) For definitions of uppercase variables, see Nag library description for subroutine

`H03ABF.(MM=M)`

```

subroutine gsetup (n, u, x, y, mx, my, mfrc, name, bkgdfile,
                 form, scale, npix, dmax, numintp, colors, overwrite,
                 showmesh, bkgdflag)
INTEGER      n, mx, my, mfrc, npix, numintp, colors(mfrc,3),
            overwrite, showmesh, bkgdflag
REAL        u(mfrc,mx,my), x(mx), y(my), scale, dmax
CHARACTER   name*10, bkgdfile*10, form*10

```

This subroutine sets up a container array for the graphics subroutines, checks to see if the container arrays are large enough, and calls the necessary subroutines for the graphics.

```

subroutine initbkgd (x, y, mx, my, mxf, myf, npix, bkgdfile,
                   bkgd, showmesh, palette, length, temp)
INTEGER          mx, my, mxf, myf, npix, showmesh, length, temp(length)
REAL            x(mx), y(my)
CHARACTER       bkgdfile*10, bkgd(npix*mx*npix*myf)*1, palette(768)*1

```

This subroutine initializes the background and background palette representing the battlefield terrain features. It also sets up the computational mesh if selected.

```

subroutine initgrafix (mx, my, mfr, utot, unum, MA, source,
                     nonzero)
INTEGER          mx, my, mfr, MA(mfr), source(mfr,3,mx*my),
                     nonzero(2,mx*my)
REAL            utot(mfr), unum(mfr,mx,my)

```

This subroutine calculates the computational mesh graphics for the initial frame based on the input force densities. It allocates on a one by one basis where each unit is allocated to the CMP with the largest 'number of units' value, which is then decreased by 1.

```

subroutine interpol (n, u, x, y, mx, my, mfr, mxf, myf, dmax,
                   colors, overwrite, maxn, dx, dy, numintp, form, name,
                   bkgdfile, npix, MA, source, mi, isrce, mi2, isrce2,
                   mi3, isrce3, num, uf, workspace, bkgd, hdf, palette,
                   showmesh, A, C, tempC, CH, LC, LR, RH, SLC, SLR,
                   assctU)
INTEGER          n, mx, my, mfr, mxf, myf, colors(mfr,3), overwrite,
                   maxn, numintp, npix, MA(mfr), source(mfr,3,mx*my),
                   mi(mfr), isrce(mfr,2,maxn), mi2(mfr),
                   isrce2(mfr,2,maxn), mi3(mfr), isrce3(mfr,2,maxn),
                   num, workspace(3*mx*myf), showmesh, A(maxn,maxn),
                   C(mfr,maxn), tempC(maxn), CH(maxn), LC(maxn),
                   LR(maxn), RH(maxn+1), SLC(maxn), SLR(maxn),
                   assctU(maxn+1)
REAL            u(mfr,mx,my), x(mx), y(my), dmax, dx, dy,
                   uf(1,mxf,myf)
CHARACTER       form*10, name*10, bkgdfile*10,
                   bkgd(npix*mx*npix*myf)*1, hdf(npix*mx*npix*myf)*1,
                   palette(768)*1

```

This subroutine linearly interpolates frames between calculated frames. The subroutine takes the list of transitional mesh sources, uses **distribute** to obtain a list of destinations, and then uses the assignment algorithm (**assct**) to match members of the two sets with one another. Once they are matched, it linearly interpolates between the positions of matched pairs to obtain the locations for the desired number of intermediate frames.

Note: All of the forces must be matched before any intermediate frames can be generated.

```

subroutine modgraf (form, zbase, dcont, delx, dely, scale, npix,
                   name, overwrite, bkgd, bkgdfile, showmesh, dmax,
                   numintp, mfrc, colors)
INTEGER           npix, overwrite, bkgd, showmesh, numintp, mfrc,
                   colors(mfrc,3)
REAL              zbase, dcont, delx, dely, scale, dmax
CHARACTER        form*10, name*10, bkgdfile*10

```

This subroutine enables the interactive modification of the graphics data.

```

subroutine raster (n, nm, mfrc, mxf, myf, form, name, bkgdfile,
                  npix, colors, overwrite, maxn, mi, isrce, bkgd, hdf,
                  palette, showmesh)
INTEGER           n, nm, mfrc, mxf, myf, npix, colors(mfrc,3), overwrite, maxn,
                  mi(mfrc), isrce(mfrc,2,maxn), showmesh
CHARACTER        form*10, name*10, bkgdfile*10, bkgd(npix*mxf*npix*myf)*1,
                  hdf(npix*mxf*npix*myf)*1, palette(768)*1

```

This subroutine generates the raster images for the DCOR program. The image data can be generated in either NCSA's HDF format or in ASCII format.

The calling tree for the visualization module subroutines described above is shown below, for each time step starting by a call from the DCOR subroutine **evolve**.

```

evolve
  gsetup
    gbal
    initbkgd
    initgrafix
      findmax
    grafix
      findmax
    distribute
      findmax
      calc
    interpol
      distribute
        findmax
        calc
      assct
      raster
    raster

```

## V. EXPLANATION OF NEW INPUT ENTRIES

This section explains the new graphics parameters for DCOR. Each entry contains a short descriptive comment in italics. After each comment is the variable name, exactly as used in the source code, printed in bold face, followed by the options, format, and a complete description of its function.

*Graphics format (none, ascii, hdf, or plot)*

**form:** This is the format of the graphics output. None produces no graphics; ascii produces a series of files (one file per frame) containing bitmap information in ASCII format; hdf produces a file containing the entire animation sequence in HDF format; and plot produces contour plots using DISSPLA as described in Ref. 9.

The remaining parameters are only required for the ASCII or HDF format options. While they must still be included in the input sequence, they will have no effect if none or plot is the selected graphics format.

*scale factor between coarse mesh and transitional mesh:*

**scale:** This real parameter determines the size of the transitional mesh. It is determined as follows:

$$\begin{aligned} \text{mxf} &= \text{int}((\text{mx}-1)*\text{scale})+1 \\ \text{myf} &= \text{int}((\text{my}-1)*\text{scale})+1 \end{aligned}$$

Once set, this parameter should not be changed interactively during an HDF animation. If it must be changed, then either the output file name should be changed or the overwrite switch described below should be set to 1.

*size of objects to be generated (in square pixels)*

**npix:** This integer parameter determines the size in pixels of the objects to be generated. Currently, each unit is represented by a square that is  $\text{int}(\text{npix}*.8) \times \text{int}(\text{npix}*.8)$  pixels in size. If npix is greater than three, then this allows up to four forces to be overlapped on the same TMP and still be seen by placing the units at different corners of the npix x npix square. This is also the relationship between the fine mesh and the bitmap. The bitmap will be  $(\text{npix}*\text{mxf}) \times (\text{npix}*\text{myf})$  pixels in size. Once set, this parameter should not be changed interactively during an HDF animation. If it must be changed, then either the output filename should be changed or the overwrite switch described below should be set to 1.

*name of graphics file(s) to be created (up to 10 chars)*

**name:** This is the root name, up to ten characters, of the generated output file. If form is set to hdf, the graphics output file will be <name>.hdf, if ascii, <name>##.#.dat, where ## is the number of the calculated frame and # is the number of the interpolated frame.

*overwrite file if it exists? (0=no, 1=yes)*

**overwrite:** If hdf is selected and this integer parameter is set to 1, then a new HDF file is started, otherwise, each successive frame will be appended to the HDF file if it exists.

*display computational mesh? (0=no, 1=yes)*

**showmesh:** This integer parameter turns on (1) and off (0) the display of the computational mesh in the animation sequence.

*name of file containing background bitmap (up to 10 chars)*

**bkgdfile:** This is the full name of the file containing the background in ASCII format (see Appendix C). Use 'none' for a white background. If the background is changed within an animation and a different palette is included, then either the output filename should be changed or the overwrite switch should be set to 1.

*maximum distance units permitted to travel between frames*

**dmax:** This real parameter is the maximum distance to be used in the linear programming algorithms.

*number of frames to be interpolated*

**numintp:** This integer parameter is the number of frames to be interpolated between each pair of calculated frames.

*RGB values for each force*

**colors(1,1):** This is the red content for the color of force 1.

**colors(1,2):** This is the green content for the color of force 1.

**colors(1,3):** This is the blue content for the color of force 1.

·  
·  
·

**colors(mfrc,1):** This is the red content for the color of force  
mfrc.

**colors(mfrc,2):** This is the green content for the color of force  
mfrc.

**colors(mfrc,3):** This is the blue content for the color of force  
mfrc.

Each colors value is an integer value ranging from 0 to 255.

## VI. CONCLUSION

We have developed, implemented, and tested a procedure for generating a graphical animation sequence from the numerical output of the PDE solver of the DCOR code. This new processing of the data allows for quick and easy display and interpretation of the battle modeling performed by the DCOR code, thus greatly enhancing DCOR's value as a combat analysis tool. Future developments and improvements for DCOR's visualization module might include development of faster linear programming algorithms, optional display of attrited units, optional display of reference grids, and optional display of plots depicting aggregate information, such as percentage of remaining forces, etc.

## REFERENCES

1. V. Protopopescu, R. T. Santoro, J. Dockery, R. L. Cox and J. M. Barnes, "Combat Modeling with Partial Differential Equations," ORNL/TM-10636, Oak Ridge National Laboratory, November 1987.
2. V. Protopopescu, R. T. Santoro and J. Dockery, "Combat Modeling with Partial Differential Equations," *Eur. J. Oper. Res.* **38**, 178 (1989).
3. P. Rusu, "Two-Dimensional Combat Modeling with Partial Differential Equations," ORNL/TM-10973, Oak Ridge National Laboratory, December 1988.
4. R. T. Santoro, P. Rusu and J. M. Barnes, "Mathematical Descriptions of Offensive Combat Maneuvers," ORNL/TM-11000, Oak Ridge National Laboratory, January 1989.
5. V. Protopopescu, R. T. Santoro, R. Cox, P. Rusu and J. Dockery, "Combat Modeling with Partial Differential Equations: The Bidimensional Case," ORNL/TM-11343 (1989).
6. F. W. Lanchester, "Aircraft in Warfare, The Dawn of the Fourth Arm--No. V., The Principle of Concentration," *Engineering* **98**,422 (1914).
7. J. G. Taylor, *Lanchester Models of Warfare*, Vols. 1 and 2, MAS, Operation Research Society of America, Virginia (1983).
8. D. Willard, "Lanchester Attrition of Interpreting Forces," *Naval Research Logistics* **37**, 31 (1990).
9. Y. Y. Azmy, "DCOR: A Deterministic Combat Model Code," ORNL/TM-11690, Oak Ridge National Laboratory, April 1991.
10. *NCSA HDF Calling Interfaces and Utilities Version 3.1*, University of Illinois at Urbana-Champaign, July 1990.
11. J. M. Hyman, "The Method of Lines Solution of Partial Differential Equations," Courant Institute of Mathematical Sciences, Mathematics and Computing Laboratory Report COO-3077-139 (1976).
12. A. C. Hindmarsh, "GEARB: Solution of Ordinary Differential Equations Having Banded Jacobian," Lawrence Livermore Laboratory Report UCID-30059, Rev. 1 (1975).
13. F. S. Hillier and G. J. Lieberman, *Operations Research*, Holden-Day, Inc., San Francisco, (1974).

14. G. Carpaneto and P. Toth, "Algorithm 548 Solution of the Assignment Problem," *ACM Transactions on Mathematical Software* **6**, 1 (1980).



APPENDIX A  
Obtaining NCSA Software  
(XImage and HDF Library)

NCSA public domain software can be obtained from NCSA through anonymous File Transfer Protocol (FTP) on the Internet.

To access NCSA's server, connect to a computer with Internet access and FTP capabilities. Then invoke FTP by typing the following:

```
ftp 141.142.20.50
```

You will be prompted for a user name. Enter 'anonymous'. Then, you will be prompted for a password. Use your electronic address as your password. You should now be logged in to the server. The command 'dir' will list all of the directories and files in your current directory. There should be several README files in this directory. It is advisable to read any README files before transferring any files. To obtain the README files, use 'mget README\*'. You may be prompted for a 'y' or 'n' to indicate whether or not you wish to download each file. These files should have descriptions of the software available and explain in greater detail the process of downloading the software.

To obtain XImage, type 'cd XImage' to change to the XImage directory. There should be a directory called 'bin' contain precompiled versions of XImage. To obtain a precompiled version of XImage, type 'cd bin' to change to the bin directory and then 'dir' to see what files are available. Next, use 'binary' to change the transfer mode to binary. Finally, type 'get <filename>' to get the desired executable. If there is not a precompiled version of XImage available, the source files are stored in a compressed format in the directory 'src'. Change to the src directory, use 'dir' to get the entire filename, (The filename changes as the version is updated.), set the transfer mode to binary, and transfer the file with 'get <filename>'. XImage requires a workstation running a XWindows compatible windowing system.

To obtain the HDF library, change to the HDF directory with 'cd HDF'. Download the README files located there and follow their instructions.

For more information regarding the use of FTP and FTP commands, refer to the manuals of your computer or ask your system administrator for help.



## APPENDIX B

### Format of the ASCII bitmap files

This is the current format of the ASCII output bitmap files. It should be easily modified to adjust to specifications of graphics packages with an importing feature.

```
nrows  ncols
max     min
1 2 3 ... nrows
1 2 3 ... ncols
nrows x ncols integer bitmap array
```

where

```
nrows = the number of rows in the bitmap array
ncols = the number of columns in the bitmap array
max = 255 = the maximum value in the bitmap array
min = 0 = the minimum value in the bitmap array
1 2 3 ... nrows = row titles
1 2 3 ... ncols = column titles
```

Staggered Defense Scenario (Janus Comparison Case)  
July 24, 1991

Final time level of simulation: 15						
Run #	# of Calc. Frames	# of Inter. Frames	Total # of Frames	CPU Time (sec.)	Real Time (sec.)	CPU Time/Frame
1	16	0	16	124.78	432.0	7.80
2	16	45	61	139.71	2592.3	2.29
3	31	0	31	129.89	3939.3	4.19
4	31	30	61	140.80	4005.1	2.31
5	61	0	61	140.18	524.3	2.30

Total CPU seconds time spent in subroutine:									
Run #	ASSCT	DIFFH	DIFFV	F	FIND-MAX	GRAFFIX	INIT-BKGD	PDETWO	RASTER
1	0.000	0.469	0.432	96.571	0.098	0.158	0.094	4.095	4.902
2	0.888	0.470	0.432	96.573	0.098	0.158	0.094	4.098	18.851
3	0.000	0.470	0.432	96.572	0.187	0.294	0.094	4.097	9.525
4	1.512	0.469	0.432	96.561	0.187	0.294	0.094	4.095	18.852
5	0.000	0.469	0.432	96.562	0.366	0.586	0.094	4.095	18.853

Number of times subroutine is called									
Run #	ASSCT	DIFFH	DIFFV	F	FIND-MAX	GRAFFIX	INIT-BKGD	PDETWO	RASTER
1	0	140995	129750	101205	2163	15	1	865	16
2	60	140995	129750	101205	2163	15	1	865	61
3	0	140995	129750	101205	4074	30	1	865	31
4	120	140995	129750	101205	4074	30	1	865	61
5	0	140995	129750	101205	7910	60	1	865	61

## APPENDIX C

### Format of the ASCII Background File

This is the format of the ASCII input background file. The subroutine *initbkgd* can easily be modified to accept other graphics formats.

```
ncols nrows ncolors
(if ncolors > 0)
  i(1)      R(1)      G(1)      B(1)
  .         .         .         .
  .         .         .         .
  .         .         .         .
i(ncolors) R(ncolors) G(ncolors) B(ncolors)
ncols x nrows integer array containing the background bitmap
```

where

ncols = the number of columns in the bitmap array  
nrows = the number of rows in the bitmap array  
ncolors = the number of colors (0-256)  
i(1) = the first color value (1-256)  
R(1) = the red content of color i(1)  
G(1) = the green content of color i(1)  
B(1) = the blue content of color i(1)  
. .  
. .  
. .  
i(ncolors) = the last color value (1-256)  
R(ncolors) = the red content of color i(ncolors)  
G(ncolors) = the green content of color i(ncolors)  
B(ncolors) = the blue content of color i(ncolors)

The number of colors should be an integer value ranging from 0 to 256 and the color values should be integers ranging from 1 to 256. The red, green, and blue elements and the data elements of the array should be integer values ranging from 0 to 255.



APPENDIX D.  
PERFORMANCE DATA  
Demonstration Simulation  
July 22-24, 1991

Final time level of simulation: 64						
Run #	# of Calc. Frames	# of Inter. Frames	Total # of Frames	CPU Time (sec.)	Real Time (sec.)	CPU Time/Frame
1	65	0	65	418.34	2821.9	6.44
2	65	192	257	507.94	9452.3	1.98
3	129	0	129	447.55	2045.8	3.47
4	129	128	257	514.93	6490.0	2.00
5	33	224	257	503.37	3505.7	1.96
6	257	0	257	508.11	7469.9	1.98

Total CPU seconds spent in subroutine:									
Run #	ASSCT	DIFFH	DIFFV	F	FIND-MAX	GRAFFIX	INIT-BKGD	PDETWO	RASTER
1	0.000	1.277	1.297	318.858	0.526	1.083	3.179	12.511	26.715
2	7.398	1.275	1.295	318.805	0.526	1.802	3.175	12.496	108.655
3	0.000	1.275	1.295	318.775	1.043	1.949	3.173	12.495	53.517
4	11.753	1.275	1.295	318.806	1.043	1.950	3.173	12.498	108.654
5	4.146	1.276	1.296	318.837	0.267	0.575	3.174	12.504	108.658
6	0.000	1.277	1.296	318.829	2.078	3.705	3.172	12.505	108.658

Number of times subroutine is called									
Run #	ASSCT	DIFFH	DIFFV	F	FIND-MAX	GRAFFIX	INIT-BKGD	PDETWO	RASTER
1	0	383232	389220	359280	10524	64	1	1996	65
2	256	383232	389220	359280	10524	64	1	1996	257
3	0	383232	389220	359280	20736	128	1	1996	129
4	512	383232	389220	359280	20736	128	1	1996	257
5	128	383232	389220	359280	5419	32	1	1996	257
6	0	383232	389220	359280	41181	256	1	1996	257



**APPENDIX E**  
**Example Input Files for Demonstration Simulation**

**Standard Input File**

```

1          Use input file wari
1          1. Start a new time batch
20         Number of time steps
20         Final time level for this batch
r         Return to execution
1          2. Start a new time batch
2         Number of time steps
22        Final time level for this batch
p         Modify diffusion & convection parameters
c         Modify velocity components
b         Modify both x- & y-components of force 1
n         non-full representation of x-component
-.25      magnitude
12 13 9 9 mesh location
0.        end x-component
n         non-full representation of y-component
.25      magnitude
4 4 5 6  mesh location
0.        end y-component
b         Modify both x- & y-components of force 2
n         non-full representation of x-component
-.25      magnitude
12 13 9 9 mesh location
0.        end x-component
n         non-full representation of y-component
.25      magnitude
4 4 5 6  mesh location
0.        end y-component
a         Do not modify force 3
a         Do not modify force 4
r         Return to execution
1          3. Start a new time batch
6         Number of time steps
28        Final time level for this batch
a         Modify attrition
n         Modify non-local attrition of force 1
0. 3. 8.
0. 3. 8.
-.00034 0. 2.
-.00034 0. 1.

```

n	Modify non-local attrition of force 2
0. 3. 8.	
0. 3. 8.	
-.00019 0. 2.	
-.00004 0. 1.	
n	Modify non-local attrition of force 3
-.00055 0. 2.	
-.00065 0. 3.	
0. 3. 8.	
0. 3. 8.	
n	Modify non-local attrition of force 4
-.00055 0. 2.	
-.00050 0. 3.	
0. 3. 8.	
0. 3. 8.	
r	Return to execution
1	4. Start a new time batch
4	Number of time steps
32	Final time level for this batch
a	Modify attrition
n	Modify non-local attrition of force 1
0. 3. 8.	
0. 3. 8.	
-.00028 0. 2.	
-.00028 0. 1.	
n	Modify non-local attrition of force 2
0. 3. 8.	
0. 3. 8.	
-.00014 0. 2.	
-.00003 0. 1.	
r	Return to menu
p	Modify diffusion & convection parameters
c	Modify velocity components (zero out force 3&4)
a	Do not modify force 1
a	Do not modify force 2
b	Modify both x- & y-components of force 3
f	full representation of x-component
180*0.	
f	full representation of y-component
180*0.	
b	Modify both x- & y-components of force 4
f	full representation of x-component
180*0.	
f	full representation of y-component

180\*0.

p

Modify diffusion & convection parameters

c

Modify velocity components

a

Do not modify force 1

a

Do not modify force 2

b

Modify both x- & y-components of force 3

n

non-full representation of x-component

-.25

3 4 11 11

-.25

5 6 10 10

-.25

6 7 9 9

-.25

7 10 8 8

0.

n

non-full representation of y-component

.25

4 4 10 10

.25

5 6 9 9

.25

6 7 7 8

0.

b

Modify both x- & y-components of force 4

n

non-full representation of x-component

-.25

3 4 11 11

-.25

5 6 10 10

-.25

6 7 9 9

-.25

7 10 8 8

0.

n

non-full representation of y-component

.25

4 4 10 10

.25

5 6 9 9

.25

6 7 7 8

0.

r

Return to execution

```

1      5. Start a new time batch
4      Number of time steps
36     Final time level for this batch
p      Modify diffusion & convection parameters
c      Modify velocity components
x      Modify x-component of force 1
n      non-full representation of x-component
-.25
7 11 9 9
0.
x      Modify x-component of force 2
n      non-full representation of x-component
-.25
7 11 9 9
0.
a      Do not modify force 3
a      Do not modify force 4
r      Return to execution
1      6. Start a new time batch
28     Number of time steps
64     Final time level for this batch
a      Modify attrition
n      Modify non-local attrition of force 1
0.  3. 8.
0.  3. 8.
-.00023 0. 2.
-.00023 0. 1.
n      Modify non-local attrition of force 2
0.  3. 8.
0.  3. 8.
-.00012 0. 2.
-.00002 0. 1.
r      Return to menu
p      Modify diffusion & convection parameters
c      Modify velocity components
b      Modify both x- & y-components of force 1
n      non-full representation of x-component
.25
4 4 7 7
.25
5 5 8 8
-.25
6 6 9 9
-.25

```

5 5 10 10

-.25

4 4 11 11

0.

n

non-full representation of y-component

.25

5 5 7 7

.25

6 6 8 8

.25

5 5 9 9

.25

4 4 10 10

0.

b

Modify both x- & y-components of force 1  
non-full representation of x-component

n

.25

4 4 7 7

.25

5 5 8 8

-.25

6 6 9 9

-.25

5 5 10 10

-.25

4 4 11 11

0.

n

non-full representation of y-component

.25

5 5 7 7

.25

6 6 8 8

.25

5 5 9 9

.25

4 4 10 10

0.

a

Do not modify force 3

a

Do not modify force 4

r

Return to execution

0

Terminate execution

**Input File wari**

numbers of x-, y-mesh points, and forces:

15 12 4

PDE solver parameters:

2 3 5

(DEMONSTRATION SIMULATION)\$

uniform x-mesh

0. 7.0

uniform y-mesh

0. 5.5

distribution of forces among sides

1 1 2 2

initial force distribution

non-full force 1

112.0

9 9 3 3

112.0

10 10 4 4

0.

non-full force 2

28.0

9 9 3 3

28.0

10 10 4 4

0.

non-full force 3

160.0

11 12 11 11

0.

non-full force 4

80.0

11 12 11 11

0.

diffusion coefficient

4\*1.e-7

convection velocity components

non-full x-component for force 1

-.25

5 5 4 4

-.25

6 9 3 3

.25

10 11 4 4

.25  
12 12 5 5  
0.  
non-full x-component for force 2  
-.25  
5 5 4 4  
-.25  
6 9 3 3  
.25  
10 11 4 4  
.25  
12 12 5 5  
0.  
non-full x-component for force 3  
-.25  
7 12 9 10  
-.04  
8 10 8 8  
0.  
non-full x-component for force 4  
-.25  
7 12 9 10  
-.04  
8 10 8 8  
0.  
non-full y-component for force 1  
.25  
4 4 4 4  
.25  
5 5 3 3  
.25  
12 12 4 4  
.25  
13 13 5 8  
0.  
non-full y-component for force 2  
.25  
4 4 4 4  
.25  
5 5 3 3  
.25  
12 12 4 4  
.25  
13 13 5 8

0.  
 non-full y-component for force 3  
 -.25  
 11 12 11 11  
 -.25  
 6 10 9 10  
 -.25  
 6 7 8 8  
 0.  
 non-full y-component for force 4  
 -.25  
 11 12 11 11  
 -.25  
 6 10 9 10  
 -.25  
 6 7 8 8  
 0.  
 non-full external source for force 1  
 0.  
 non-full external source for force 2  
 0.  
 non-full external source for force 3  
 0.  
 non-full external source for force 4  
 0.  
 local linear interactions for force 1  
 4\*0.  
 local linear interactions for force 2  
 4\*0.  
 local linear interactions for force 3  
 4\*0.  
 local linear interactions for force 4  
 4\*0.  
 local quadratic interactions for force 1  
 16\*0.  
 local quadratic interactions for force 2  
 16\*0.  
 local quadratic interactions for force 3  
 16\*0.  
 local quadratic interactions for force 4  
 16\*0.  
 nonlocal interactions for force 1  
 0. 3. 8.  
 0. 3. 8.

```
0. .1 2.5
0. .1 1.
nonlocal interactions for force 2
0. 3. 8.
0. 3. 8.
0. .1 2.5
0. .1 1.
nonlocal interactions for force 3
0. .1 2.5
0. .1 3.5
0. 3. 8.
0. 3. 8.
nonlocal interactions for force 4
0. .1 2.5
0. .1 3.5
0. 3. 8.
0. 3. 8.
BCs for force 1 (top,right,bottom,left)
0. 1. 0.
0. 1. 0.
0. 1. 0.
0. 1. 0.
BCs for force 2 (top,right,bottom,left)
0. 1. 0.
0. 1. 0.
0. 1. 0.
0. 1. 0.
BCs for force 3 (top,right,bottom,left)
0. 1. 0.
0. 1. 0.
0. 1. 0.
0. 1. 0.
BCs for force 4 (top,right,bottom,left)
0. 1. 0.
0. 1. 0.
0. 1. 0.
0. 1. 0.
Graphics format (none, ascii, hdf, or plot)
hdf
contour parameters: base value, increment
-150. 151.
x- and y-axis increments
2. 2.
scale factor
```

8.  
size of objects to be generated(in square pixels)  
5  
name of graphics file(s) to be created (up to 10 chars)  
demo  
overwrite file if it exists? (0=no, 1=yes)  
1  
display computational mesh? (0=no, 1=yes)  
1  
name of file containing background bitmap (up to 10 chars)  
Desert.dat  
maximum distance to be traveled between images  
2.  
number of frames to be interpolated  
0  
RGB values for each force  
255 0 0  
0 255 0  
0 0 255  
255 255 0

**APPENDIX F**  
**Example Frames from Demonstration Simulation Run #1**

Figure F.1 -- Frame # 1: Initial frame of Demonstration Simulation.

Figure F.2 -- Frame #21: Red and Green forces maneuver around mountain.

Figure F.3 -- Frame #23: Red and Green forces move to attack position.

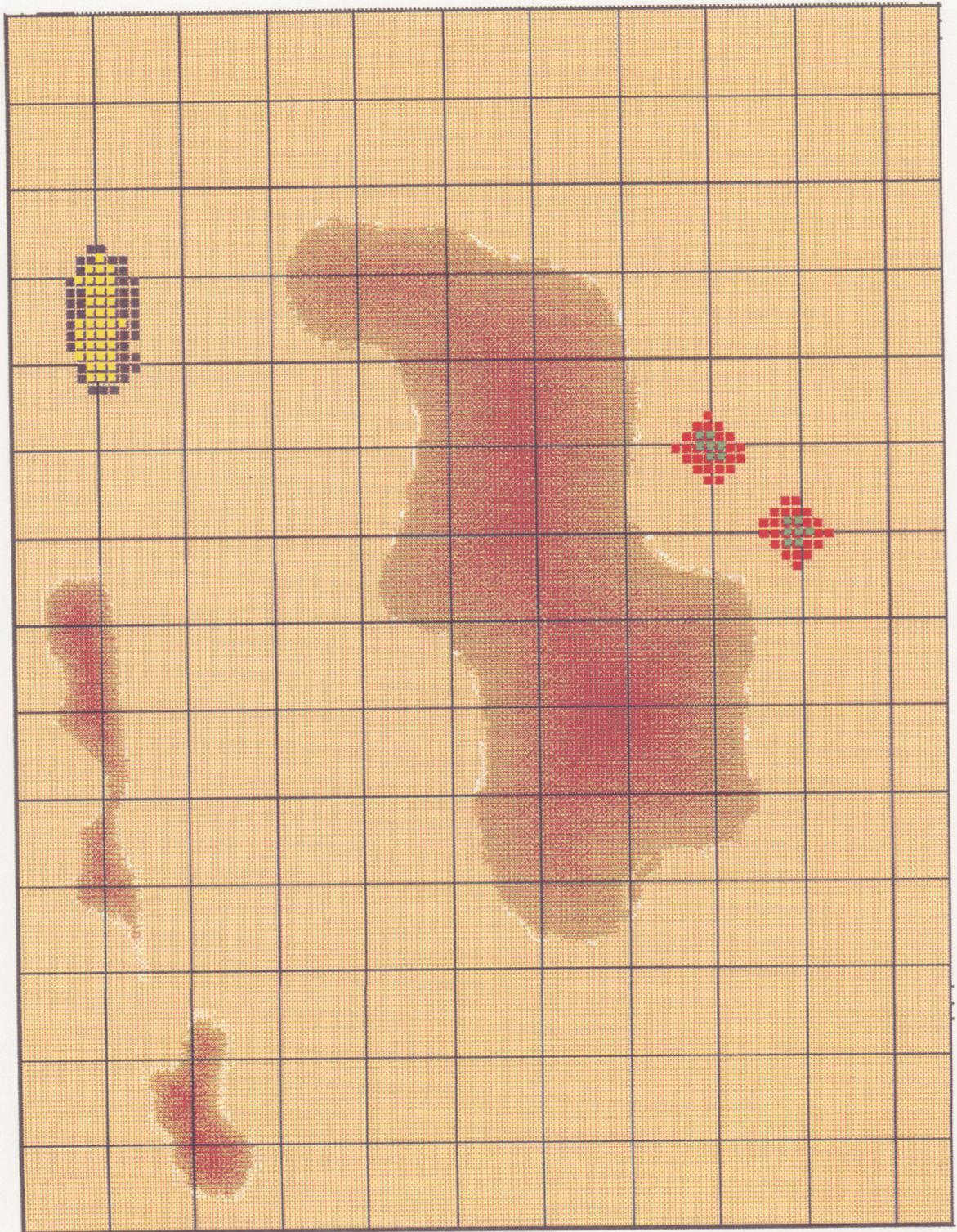
Figure F.4 -- Frame #29: Red and Green forces start attacking Blue and Yellow forces.  
Attrition begins.

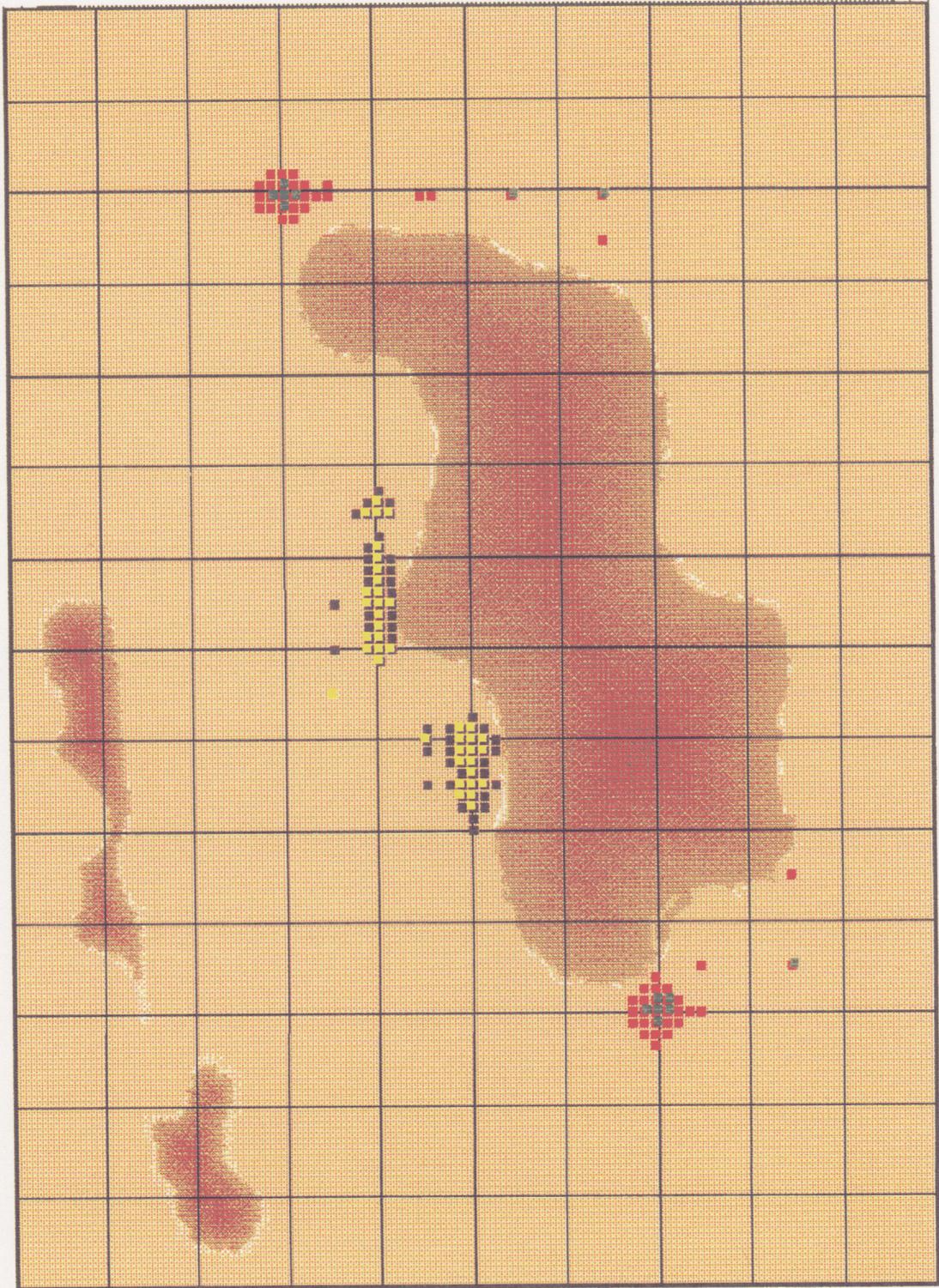
Figure F.5 -- Frame #33: Blue and Yellow forces begin to retreat.

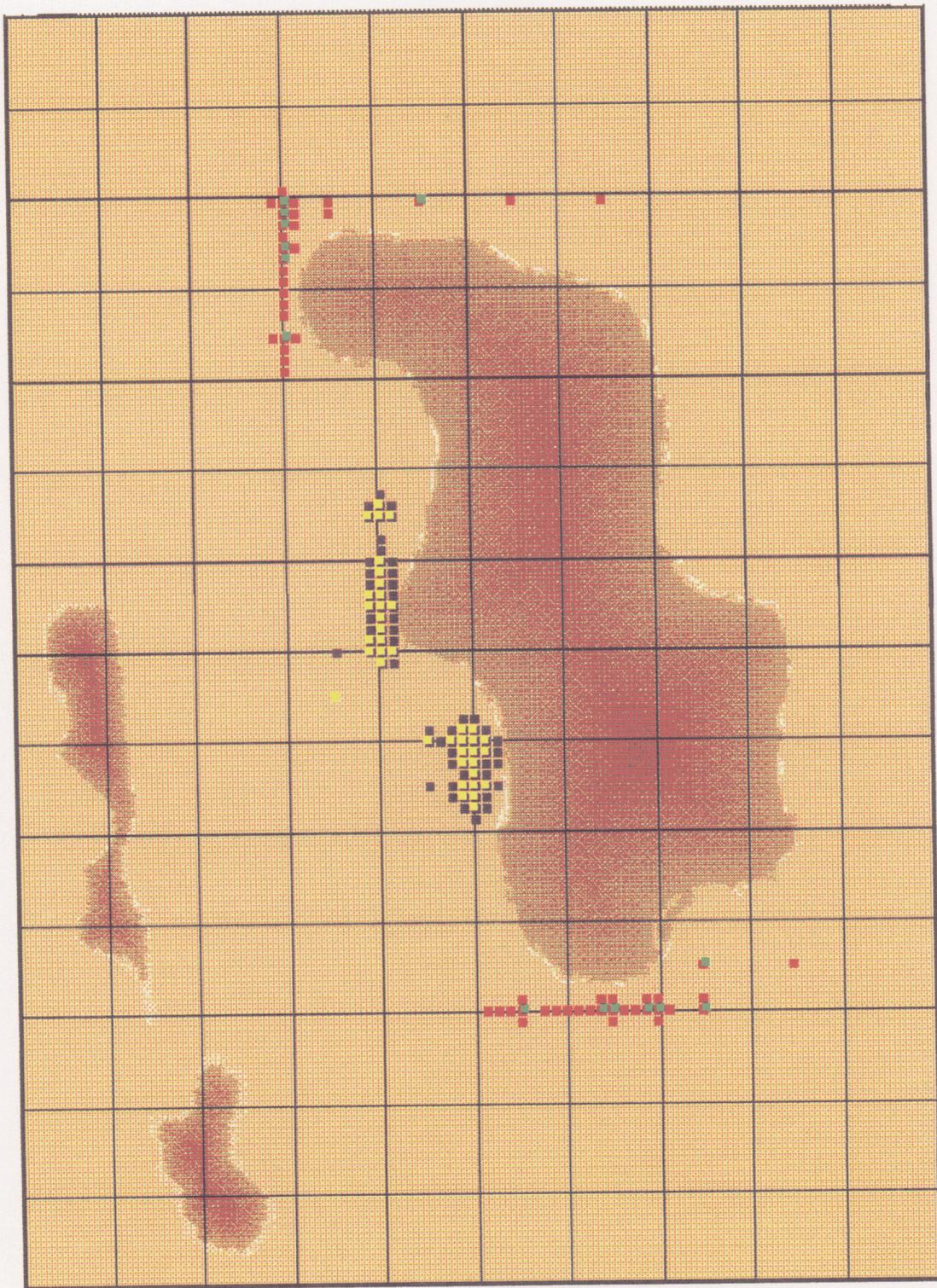
Figure F.6 -- Frame #37: Red and Green forces begin pursuit.

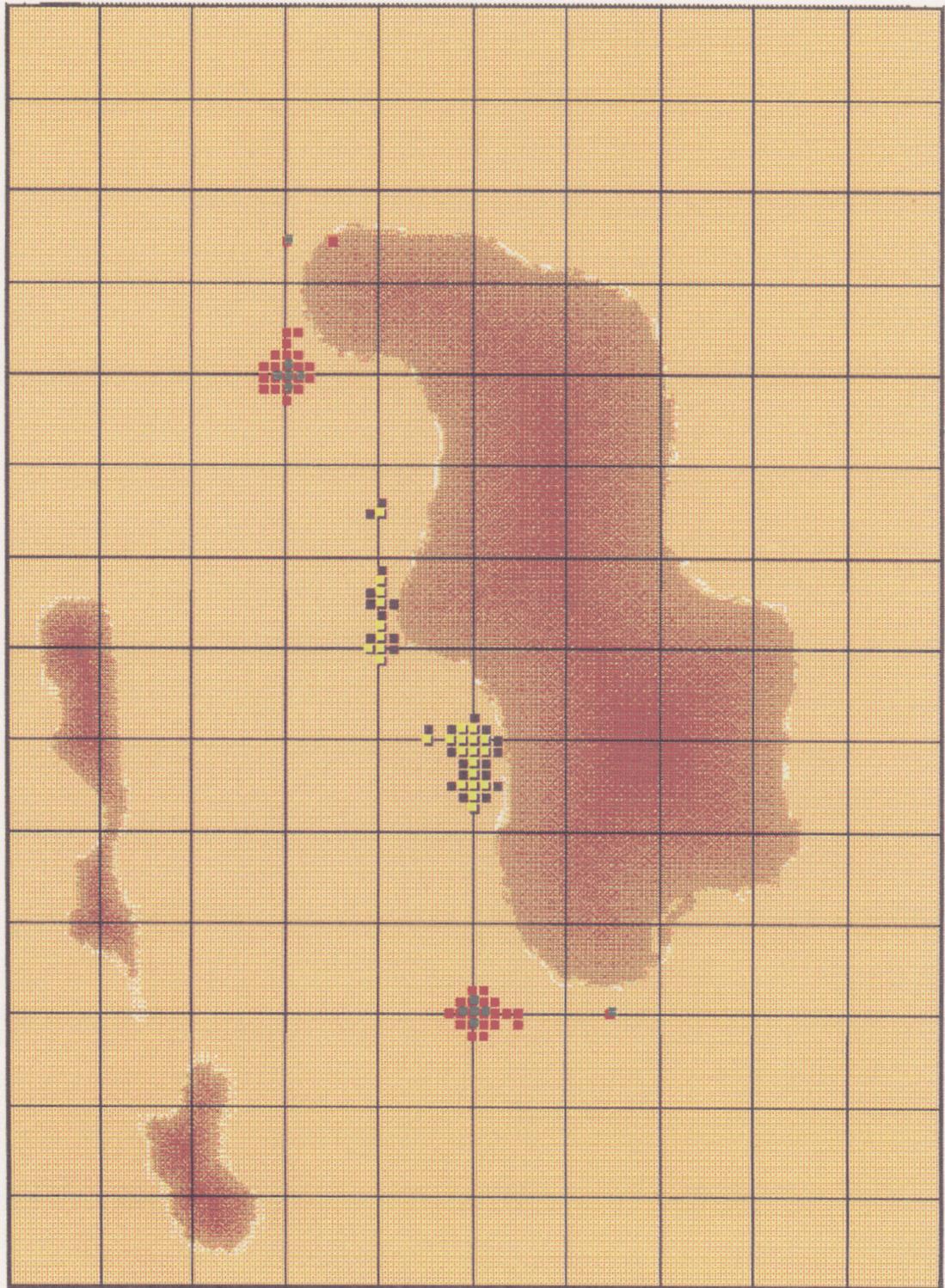
Figure F.7 -- Frame #65: Blue and Yellow forces are almost annihilated. Red and Green victory.

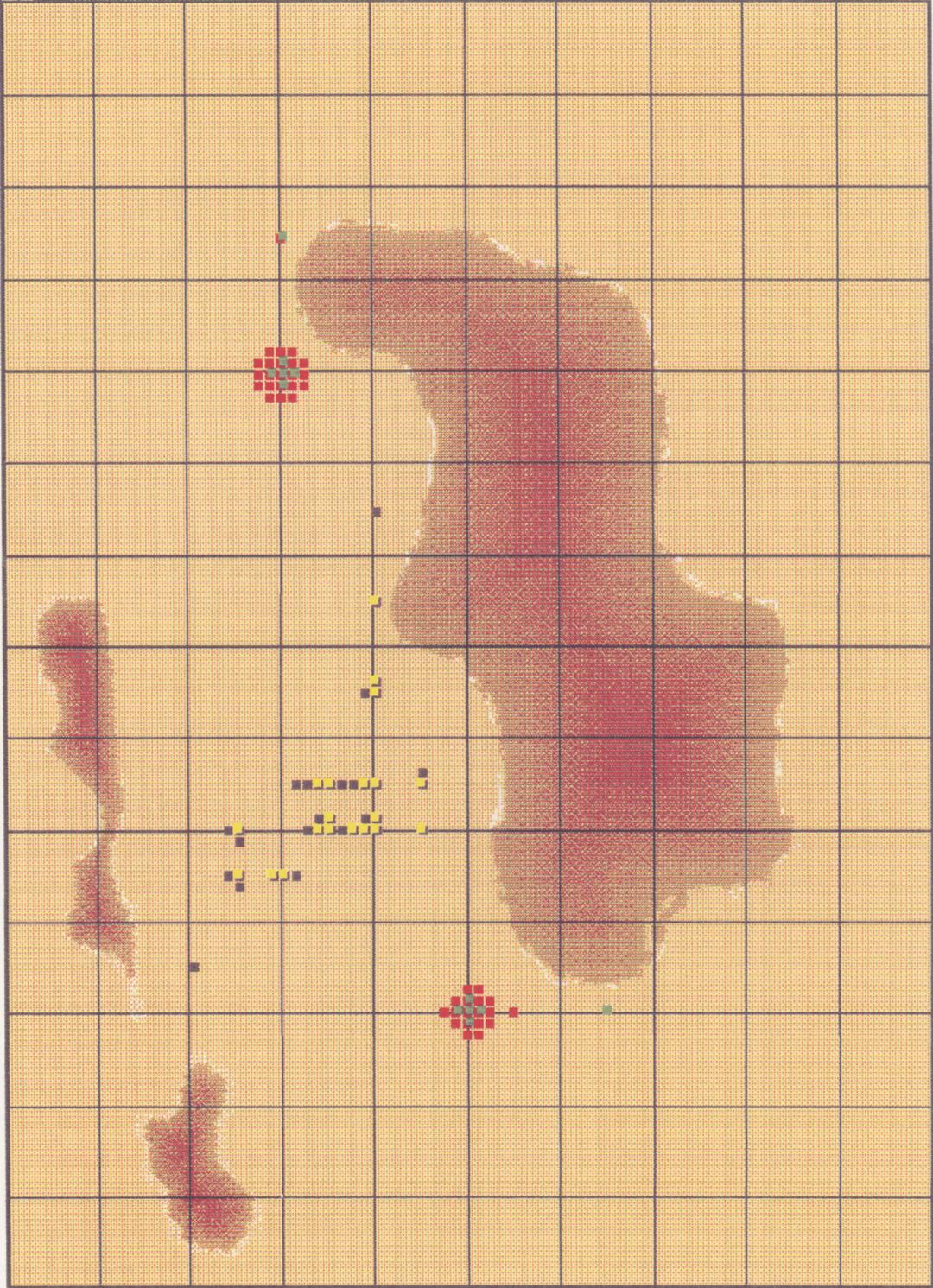


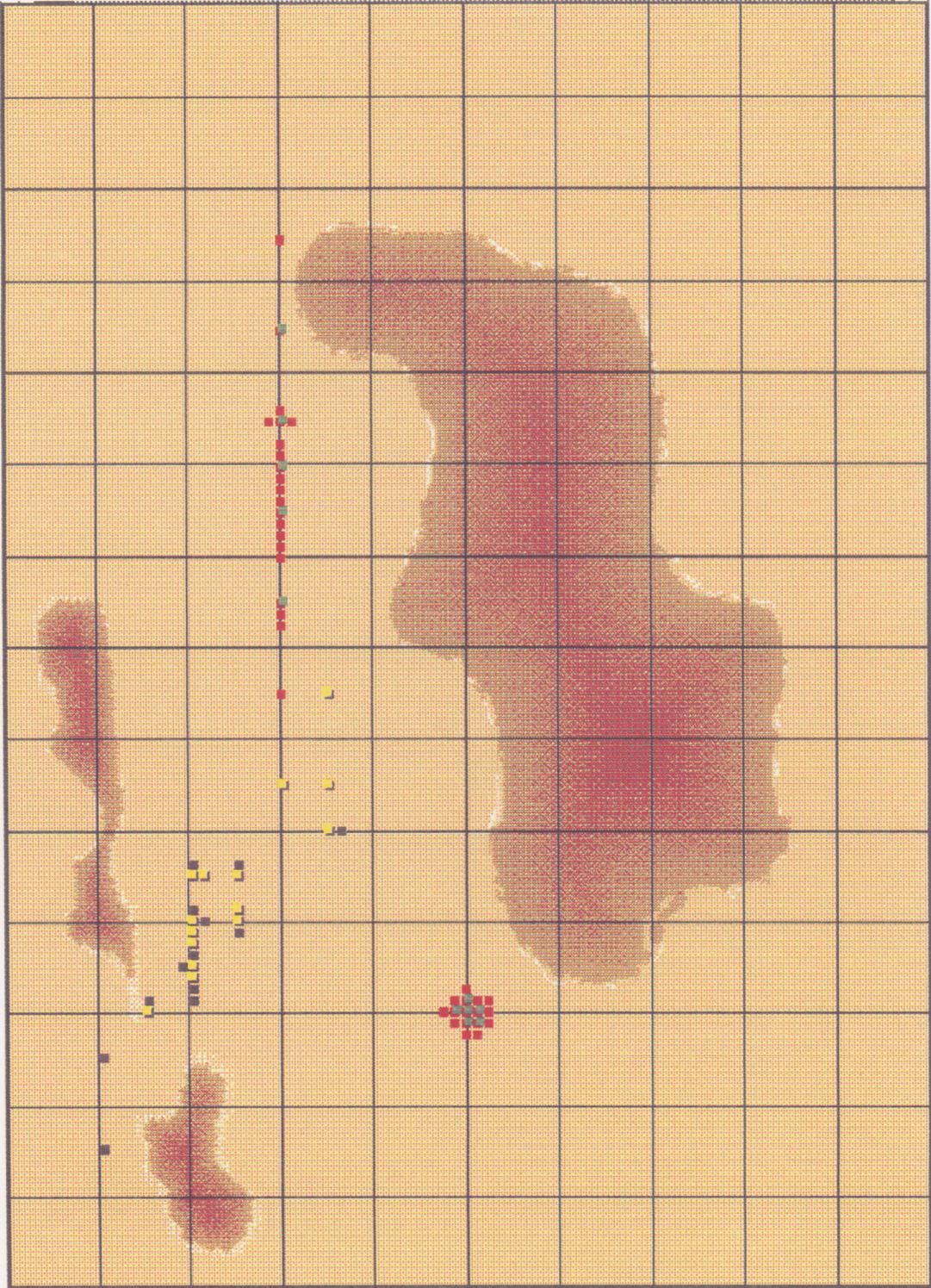


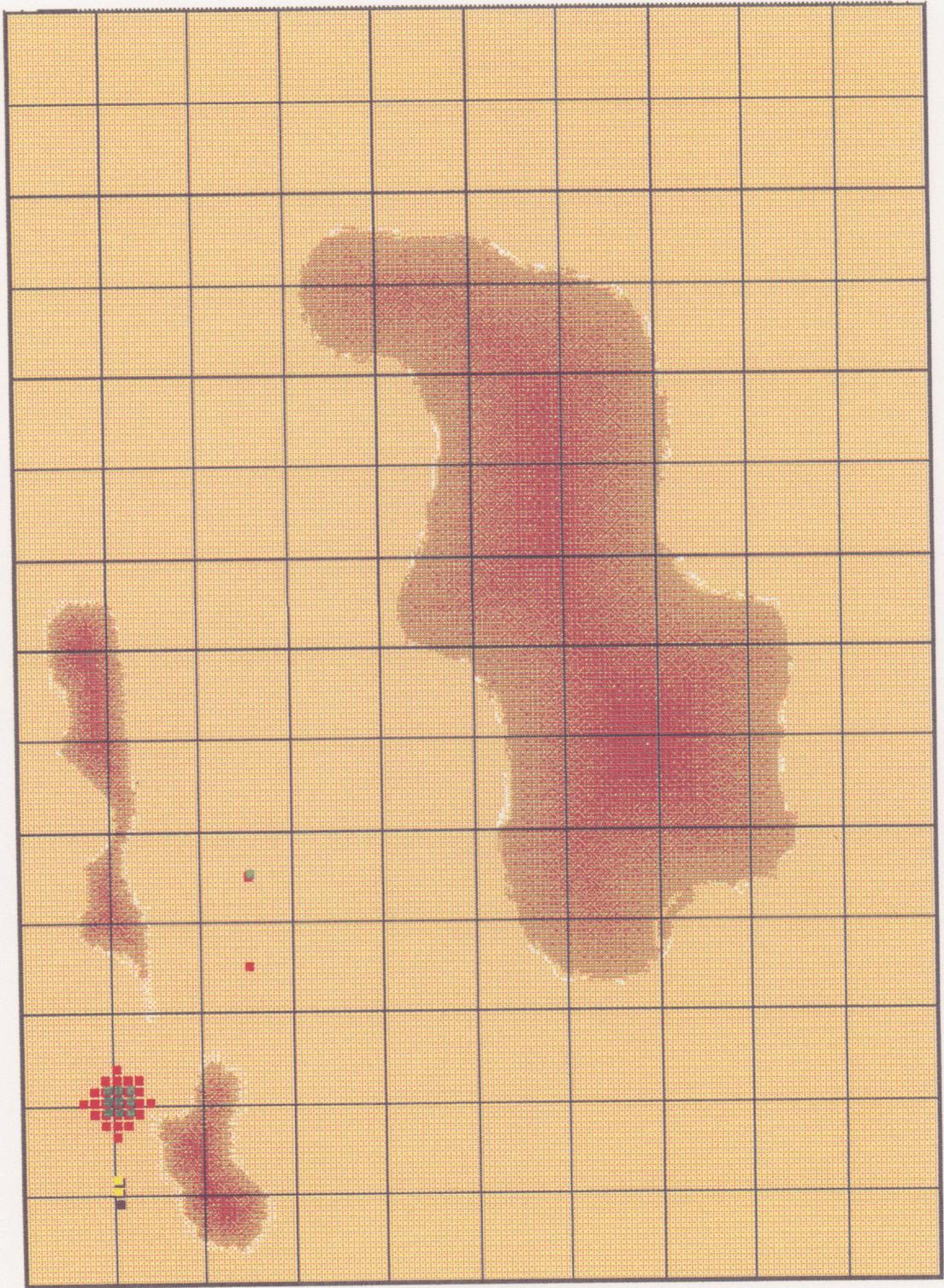












## INTERNAL DISTRIBUTION

- |        |                    |        |                                  |
|--------|--------------------|--------|----------------------------------|
| 1.     | B. R. Appleton     | 19.    | R. C. Ward                       |
| 2-6.   | Y. Y. Azmy         | 20.    | EPMD Reports Office              |
| 7.     | R. M. Davis        | 21-22. | Laboratory Records<br>Department |
| 8.     | W. Fulkerson       | 23.    | Laboratory Records,<br>ORNL-RC   |
| 9.     | D. S. Hartley III  | 24.    | Document Reference<br>Section    |
| 10.    | D. T. Ingersoll    | 25.    | Central Library                  |
| 11.    | V. A. Protopopescu | 26.    | ORNL Patent Section              |
| 12-16. | R. T. Santoro      |        |                                  |
| 17.    | R. J. Toedte       |        |                                  |
| 18.    | B. S. Wallace      |        |                                  |

## EXTERNAL DISTRIBUTION

27. Prof. Roger W. Brockett, Harvard University, Pierce Hall, 29 Oxford Street, Cambridge, MA 02138
28. J. J. Dorning, Department of Nuclear Engineering and Physics, Thornton Hall, McCormick Road, University of Virginia, Charlottesville, Virginia 22901
29. Bruce W. Fowler, U. S. Army Missile Command, ATTN: AMSMI-RD-AC, Redstone Arsenal, Alabama 35898-5282
30. R. M. Haralick, Department of Electrical Engineering, University of Washington, Seattle, Washington, 98195
31. R. L. Helmbold, Concepts Analysis Agency, 8120 Woodmont Avenue, Bethesda, Maryland 20814
- 32-36. Greg Hunt, 6921 Richards Lane, Austell, Georgia 30001
37. Dr. James E. Leiss, Rt. 2, Box 142C, Broadway, VA
38. Dr. Donna Llewellyn, School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332
39. Neville Moray, Department of Mechanical and Industrial Engineering, University of Illinois, 1206 West Green Street, Urbana, Illinois 61801
40. Prof. H. Donald Ratliff, School of Industrial and Systems Engineering, Georgia Institute of Technology, Room 325, Atlanta, Georgia 30332
41. Mary F. Wheeler, Department of Mathematical Sciences, Rice University, P.O. Box 1892, Houston, Texas 77251
55. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations, P.O. Box 2001, Oak Ridge, Tennessee 37831
- 56-65. Office of Scientific and Technical Information, P.O. Box 62, Oak Ridge, Tennessee 37830