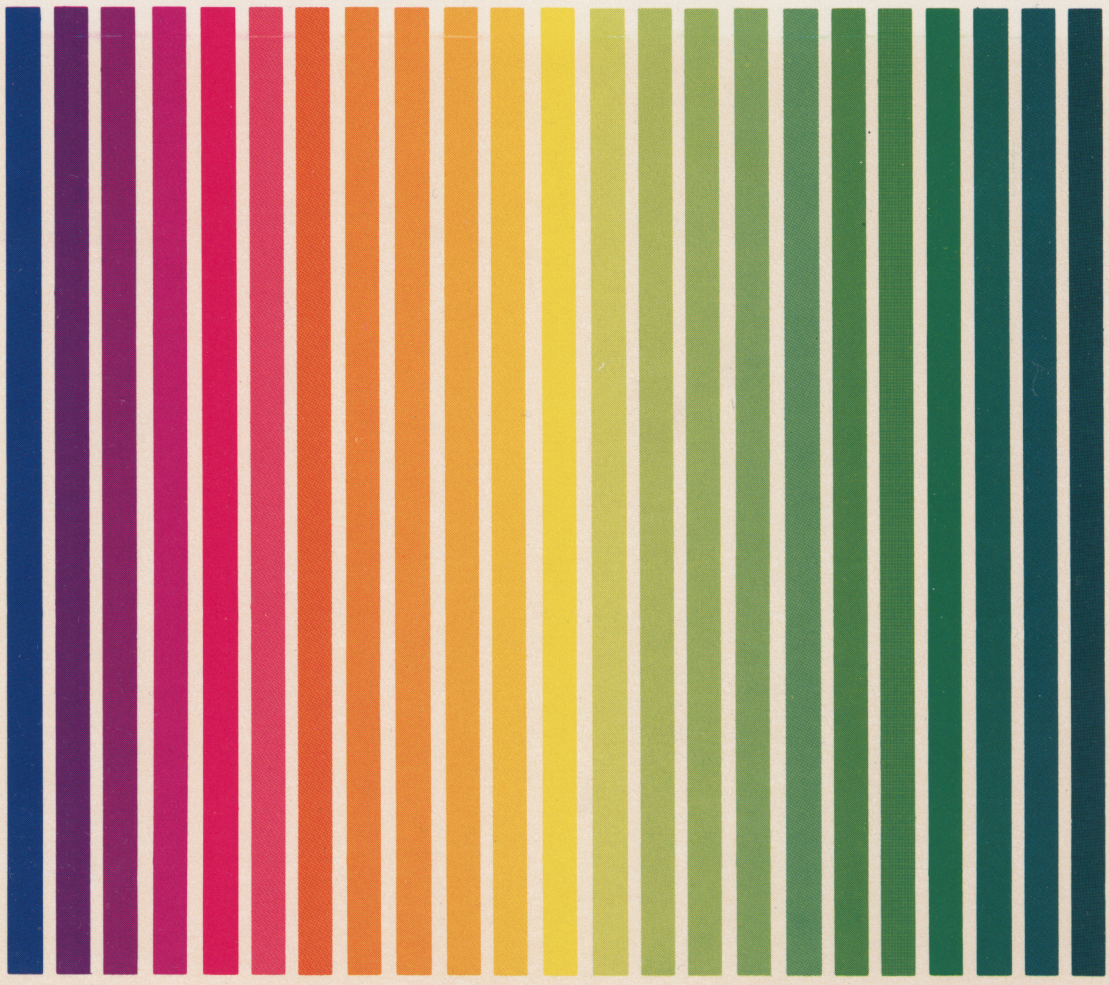
APX ATARI® PROGRAM EXCHANGE



Chris Crawford

SOURCE CODE FOR EASTERN FRONT (1941)

APX-20095

User-Written Software for ATARI Home Computers

Chris Crawford

SOURCE CODE FOR EASTERN FRONT (1941)

APX-20095

SOURCE CODE FOR EASTERN FRONT (1941)

by

Chris Crawford

TRADEMARKS OF ATARI

The following are trademarks of Atari, Inc.

ATARI
ATARI 400 Home Computer
ATARI 800 Home Computer
ATARI 410 Program Recorder
ATARI 810 Disk Drive
ATARI 820 40-Column Printer
ATARI 822 Thermal Printer
ATARI 825 80-Column Printer
ATARI 830 Acoustic Modem
ATARI 850 Interface Module

****

Distributed by

The ATARI Program Exchange
P. O. Box 427
155 Moffett Park Drive, B-1
Sunnyvale, CA 94086

To request an APX Software Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/745-5535.

****

EASTERN FRONT DOCUMENTATION PACKAGE

This package contains material of value to any programmer attempting to study the program EASTERN FRONT (1941). My purpose in making these materials available is to provide programmers with an instructive lesson in designing and programming a major game. This program demonstrates many aspects of the game designer's art: high-level design concepts, algorithms for wargames, programming structure and technique, and specific applications of the special capabilities of the ATARI Home Computer™. I cannot claim that the program is of textbook clarity; indeed, it is fraught with clumsy inanities. I made no efforts to conceal or correct the mistakes in the program. I believe that most programmers live by a double standard. They expect all code to be clean, tight, and elegant, yet they are seldom able to achieve this goal. I wanted to show this program "warts and all". I am not proud of the warts; I simply won't deny their existence. Furthermore, they are themselves instructive. By studying them, the programmer can see how mistakes are made and can better avoid them.

My hope is that people will study these materials to become better programmers with the ATARI Home Computer. There will also be smaller-minded individuals who see them not as instructional materials but as sources of profit. I'm sure some yokel will perform some trivial modifications to the code and start selling WESTERN FRONT 1944 or some similar rip-off. Modifying an existing program is a useful exercise for the beginning programmer. Selling such a program without proper authorization is not legally secure, economically realistic, or professionally respectable. If you are seriously interested in modifying EASTERN FRONT 1941 for commercial reasons, then contact me before you begin work. I will entertain proposals for extensions which do not sully the original product.

This is a very complex program; to explain completely all aspects of the program would take far too much space. I have tried to include in this package all the key items that a programmer would need to understand the program. I assume that the user of this package is already a competent programmer who is familiar with assembly language and the structure of the ATARI Home Computer. I also assume that you have played the game and understand its functions. This makes my task shorter. If you are a beginning programmer, you will not be able to understand what is in here. Even the competent programmer will find some of the quirks of this program mystifying. A few of these strange quirks are brilliant strokes of programming genius; the majority are simple mistakes.

The following items are included in this package:

EASTERN FRONT STRUCTURE

EASTERN FRONT 1941 is divided into six modules. The program was developed with the Atari Assembler/Editor cartridge, which has no linking facility. Therefore, the modules were linked by hand. This makes the program more difficult to understand and modify.

The six modules and their functions are as follows:

FONTS.DAT	a data module containing character fonts for the map
EFT18D.ASM	Data module: display list, map and troop data
EFT18I.ASM	Interrupt routines: joystick, scrolling, orders
EFT18M.ASM	Mainline: initialization, movement, seasons
EFT18C.ASM	Combat: combat and logistics routines
EFT18T.ASM	Thinking: artificial intelligence routines

The sequence above is the historical sequence in which the modules were developed. The later modules are structurally higher than the earlier ones. They frequently make use of subroutines and tables in the earlier ones while the reverse is rare.

The program was designed to run in a 16K machine with a cassette only. To achieve this goal I had to scrunch the program very tightly. The lack of good linking facilities made scrunching a difficult task. I was forced to take some subroutines and data tables out of one module and insert them into another module. Many times the positioning of a subroutine or table was decided not by logic or structure but rather by the fortuitous discovery of a chunk of space in one module that was precisely the right size to accommodate the homeless code.

Virtually all of the memory space available to the 16K system is used. There are a few unused chunks of RAM, but they are rather small. I did preserve the 1K region used by the Operating System for its Mode 0 display list and display data. This RAM could be stolen by a desperate programmer, but the Mode 0 display shown while loading the program would go wild, possibly frightening the user into unfortunate recourse to the SYSTEM RESET key. The programmer should study the global memory map on page 54 very closely before appropriating any memory. You should also refer to the appropriate source code listing. I repeat, there is very little available memory.

DATA MODULE

This is the simplest of the modules. It is nothing more than a collection of data bytes. Many inexperienced programmers think of a program in terms of the executable code. The code is only one portion of the entire program. The data is the other major component. Both components are necessary, but many programmers neglect the data. Don't make this mistake. The data needs as much attention as the code.

MILITARY STATE VARIABLES

The first data tables are the values for the military units. These are presented in a more orderly fashion in the Unit Characteristics Chart on pages 56-58. There are 159 different units recognized in this game. Of these, 54 are German and 105 are Russian. These numbers are critical; you will see them often in the code in one form or another.

The first two data tables are CORPSX and CORPSY (lines 30-330). These tables specify the initial map coordinates for the military units, corps for the Germans and armies for the Russians. The coordinate system is the same one used for the map; see the map reproduced on page 55.

The next two data tables are MSTRNG and CSTRNG (lines 340-570). These tables store the muster and combat strength of the units. The combat strength is initialized at the beginning of the game to equal the muster strength.

Next comes the SWAP table (lines 580-790). This table serves two purposes. It contains the character type of the unit (infantry or armor) for use when the unit is put onto the map. The same table also acts as a buffer to store the terrain underneath the unit. The unit's image is swapped with the terrain image, hence the label.

The table called ARRIVE (lines 800-1000) tells the turn on which each unit first arrives on the map. It is a reinforcement schedule. Note that some units are set to arrive on turn 255. As in the real world, it is sometimes more convenient to postpone beyond reasonable limits some commitment that we cannot actually refuse but no longer wish to honor. This table is frequently used to determine if a unit is on the map. Many sections of code begin with LDA ARRIVE, X/CMP #$FF/BEQ NEXT to weed out units that are either already dead or not yet on the map.

CORPT (lines 1180-1380) specifies the type of unit. There are many different types of units in this game, but only three types are recognized in the mechanics of the game: infantry, armor, and militia. I do recognize different types of units for identification purposes. There are panzergrenadier, mountain, paratroop, and SS units for the Germans and Guards, tank and shock armies for the Russians, among others. There are also the different nationalities. All these factors are encoded in the single CORPT constant.

CORPNO (lines 1390-1590) specifies the military unit number, as in the 48th Panzer Corps. This is another quantity that has no significance to the

operation of the game but is included for the unit description when a unit is examined. Such nonfunctional elements in a game are referred to as "color". I call them "dirt". My bad manners are exceeded only by my hypocrisy, for I still use such elements in my own games. Oink.

These eight parameters completely determine the state of a military unit. They were the first items I defined when I set about designing the game. By defining them at the outset, I fixed what the game would and would not be able to do. This lent focus to the design. Before doing any simulation, you must declare precisely what you know before you attempt to do anything with it.

WORDS TABLE

Another chunk of this module is devoted to the WORDS table (lines 1010-1170), which gives the text strings used in the text windows. I decided to use a fixed field size of eight characters rather than a variable field size. There are only a few cases where the words I need are too long to fit: SEPTEMBR, HUNGARAN, PARATRP, PZRGRNDR. The decision to use eight characters per field was a good one. The code to put text on the screen is fast and simple, and the data tables required are short.

CONVERTING BYTES TO DIGITS

Line 1600 begins one of the strangest ideas I have ever implemented in a program. It is also one of the stupidest. I was worried about the conversion of hexadecimal byte values in my tables into numeral strings on the screen. Whenever the player presses the button to raise a unit in the cursor, the interrupt routine must put a considerable amount of information into the text window. It must first find out which unit is in the cursor, then look up the unit's CORPNO, CORPT, MSTRNG, and CSTRNG. It must then translate all these quantities into readable text and place that text onto the text window. Furthermore, the entire operation must be completed during the 2000 machine cycles available in the vertical blank interrupt routine. These requirements impose severe time constraints on any code.

My solution was pretty ruthless. I created three tables in memory, one for the hundreds digit of a byte, one for the tens digit, and one for the ones digit. With these tables the task of hexadecimal to decimal text conversion became simple. I put the byte to be converted into the X register and LDA HDIGIT,X. That one instruction produces the hundreds digit. Similar operations with TDIGIT and ODIGIT give the other digits. The total time for conversion is 12 cycles. That's extremely fast! Unfortunately, it is also extremely RAM-expensive. Those three tables require 768 bytes.

The alternative is to calculate the conversion value rather than look it up. The following routine is a standard way to solve the problem:

```
;start with byte to be converted in accumulator
        LDX  #$FF
        SEC
LOOP1   INX
        SBC  #$64
        BCS  LOOP1
        STX  HDIGIT
        ADC  #$64
        LDX  #$FF
        SEC
LOOP2   INX
        SBC  #$0A
        BCS  LOOP2
        STX  TDIGIT
        ADC  #$0A
        STA  ODIGIT
```

This code will require at most 108 cycles to execute. Now, 108 cycles is not much machine time, but the conversion has to be done three times during vertical blank interrupt. Thus the method I chose to use saves me nearly 300 machine cycles out of 2000 available. That is why I chose a memory-wasteful algorithm.

Did I make the right decision? It is very difficult to calculate how many cycles my routine needs. I know that it consumes at least 1700 cycles in the worst case. Without a logic analyzer it is very difficult to say anything more. I might have gotten away with the standard algorithm. This discussion illustrates the nature of the guesswork that a designer must use. When you are in the early stages of writing a program, you have no way of knowing how big or how slow your code will be. You must rely on hunches. My hunch told me to trade memory for time. Such conservatism is very important in the early stages of the programming phase. Once a problem is built into a program, it is extremely difficult to expunge. Problems should be prevented before you have exhausted your reserves of memory and execution time.

MORE MISCELLANEOUS TABLES

The next table in the data module is called TXTTBL (lines 2450-2500). It is a table of long text messages. I chose a fixed field length of 32 bytes for these messages. There are only three messages here.

MONLEN (lines 2510-2520) is a table giving the lengths in days of the months. MONLEN is 13 bytes long. More astute readers may recall that this does not quite correspond with the number of months in a year. This is an example of lazy coding. I chose to number my months from 1 rather than zero. It made more sense to me. I was unwilling to hassle with the

redefinition problem arising from my inappropriate numbering system. Rather than think the problem through I decided on a brazen solution. "What the hell!", I cried, "Let's waste a byte! I've got plenty to spare!" I'm a devil-may-care rascal.

The next two tables, HMORDS and WHORDS (lines 2530-2540), keep track of the orders given to the units during the course of the game. They are initialized to zero at the beginning of the game. HMORDS tells how many valid orders are in storage, and WHORDS tells what the orders are. This game uses a rectangular grid, so each unit can move in any of four directions. It takes two bits to specify one of four orders. Thus, the two bytes of WHORDS allocated for each unit can store up to eight orders.

There is an interesting bug in EASTERN FRONT 1941 associated with these two tables. Under certain conditions HMORDS can get a value greater than eight. When this happens the arrow showing the future path of the unit keeps moving right off the edge of the map. I have never found the cause of the bug. The bug is rare and nondestructive, so I never bothered expending the time to track it down.

BEEPTB (line 2550) is a table of frequencies used to give feedback when the joystick is used to give orders.

ERRMSG (lines 2560-2630) is a table of error messages. Like the other text messages, I use a fixed field length of 32 bytes. Only four error messages are supported, yet together they consume 128 bytes of RAM. This demonstrates why textual error messages are so rare in personal computers.

The number and type of error messages are a revealing indication of the quality of human engineering in the program. The ideal program has no error messages, because it would make errors inconceivable. The four errors generated by this program could have been avoided with sufficient effort on my part. All four concern the entry of orders. The "only eight orders allowed" error could have been prevented by the simple expedient of using more bytes for storing orders. Of course, there has to be some kind of limit, and I think eight is a reasonable limit, so I can rest easy with this one. The "please wait for Maltakreuze" error was purely a matter of programmer convenience; I had problems implementing the code necessary to allow orders to be entered immediatly, so I hid behind the excuse that the user should wait to see what he has already entered before he adds new orders. Again, this is a reasonable defense. I now think that I should have sped up the arrow so that it moves faster. This would have made the error less common. The error "That is a Russian unit" could have been dispensed with. It might have been better to ignore orders given to Russian units. I don't know about this one. The last error, "no diagonal moves allowed", bothers me greatly. I could have allowed diagonal moves, simply interpreting a diagonal move as a combination of horizontal and vertical moves. However, the resolution on the joystick is so poor that many people can mistakenly enter a diagonal move when they intended to enter only a horizontal or vertical move. I am torn between protecting my user and accommodating him.

The tables in lines 2640-2680 are used for logical manipulation of the joystick entries and for unit motion.

TRTAB (lines 2690-2700) is a table of monthly colors for trees. It is the table that allows me to change the color of the trees as the seasons go by. It is only 13 bytes long. The extra byte can be attributed to my wanton disregard for the requirements of tight coding.

MLTKRZ (line 2710) is a bit map of the maltese cross.

The RAM from $6000-$63FF is reserved for the two graphics character sets. They are contained in file FONTS.DAT.

The display list comes next (lines 2780-2830). It is rather long because I reload the memory scan counter on each ANTIC mode 7 line. This is necessary for proper fine scrolling. Note also the blank lines inserted into the display list.

ARRTAB (line 2840) is a bit map of the arrows used to display existing orders. One shape is used for each of the four cardinal directions.

The screen data for the text window comes next. An interesting oddity of the text window arises from the history of the program. I originally put the date window in the main text window at the bottom of the screen. Later on I decided for aesthetic reasons to move the date window to the top of the screen. This was accomplished with a simple change in the display list. The upshot of this is that the screen data area for the date window comes after the screen data area for the text window at the bottom of the screen.

Lines 2950-5400 contain the map data. This huge chunk contains all of the terrain. It acts both as display data and as terrain behavior data. I had no need to keep separate images of the map, one for display and one for computations. The same 2K chunk fills both needs. The numbers stored here are the character codes for the ANTIC mode 7 display. The 127 code is a border character used to indicate the edge of the map. For a fuller understanding of how the map works, consult the map image figure and the character set definition.

Line 5410 gives a table called STKTAB. This table is used in decoding joystick values. You may have noticed that I use tables rather heavily. In general, table-driven solutions to programming problems are frequently more desirable than solutions implemented directly in code. They offer far greater flexibility and are normally simpler to program. Furthermore, table-driven routines normally execute faster than code-intensive routines. This point is discussed further in the comments on the interrupt module.

The TRNTAB (lines 5440-5490) specifies the number of subturns expended to enter a given type of square under given weather conditions. A wargamer would call it a movement point costs chart. An entry of 128 indicates that the square in question can never be entered. The operation of this table is a little messy. There are ten terrain types supported, with different values for each of three seasons and two unit types. Thus, there are sixty

entries in this table. Ten entries for infantry alternate with ten entries for armor. Twenty entries for summer are followed by twenty for mud and twenty for snow. The SSNCOD table on line 5430 gives an index into TRNTAB as a function of month. The terrain table is on page 63.

The four following tables (BHX1 through BHY2---lines 5500-5570) specify blocked movement paths. One of the worst problems I encountered in designing the movement algorithms of this game involved blocked movement. It is a simple matter to determine whether motion into a particular type of square, say an ocean square or a border square, is forbidden. Just look at the terrain type and you know that no unit can enter the square. However, there are unfortunate circumstances in which two legitimate squares can be inaccessible to each other. For example, consider the coastline squares of southern Finland and northern Estonia. These squares are adjacent to each other and are all land squares, so a simple-minded program would allow units to move freely from one square to the other. The only problem with this is that the Gulf of Bothnia lies between the two coastlines. Armies cannot walk on water. How can the program detect this condition?

I wrestled with a number of possible algorithms. Most of my early attempts focused on devising an intelligence that would perceive the nature of the situation and act accordingly. I tried all sorts of clever algorithms. All were big and slow. None worked reliably. The scheme I finally chose is remarkably stupid. I found only 11 pairs of squares on the map that caused this problem. I created a table of these square pairs. During movement, the program tests if the unit is attempting to move between a forbidden pair. If so, movement is denied. The table labels stand for Bad Hex X coordinate 1, Bad Hex Y coordinate 1, etc. I'm an old time wargamer and I still think in terms of hexes even though the game uses squares.

This case is an excellent example of the usefulness of table-driven solutions. Logic-driven solutions did not work acceptably, yet the table-driven solution was simple and easy to implement.

The last chunk of RAM reserved by the module is EXEC. This table holds the execution times of the units. The number stored here specifies the subturn in which the unit's next order will be executed.

INTERRUPT MODULE

This module handles all of the I/O for the game. It consists of two routines: a vertical blank interrupt routine which is executed at the beginning of each frame, and a display list interrupt routine which is executed several times during each frame. It is not possible for these two routines to operate together, or for one routine to interrupt the other. The vertical blank interrupt routine reads and responds to the joystick. It performs the scrolling, picks up units and displays the unit data, accepts orders inputs, and displays existing orders. The entire vertical blank interrupt routine must operate under tight timing requirements, as there are only 2000 machine cycles available during vertical blank.

COORDINATE SYSTEMS

The coordinate systems used by this module will drive you nuts. I must admit that I didn't quite know what I was doing as I wrote this module, so whenever I encountered a problem I simply spawned a new coordinate system to deal with it. The result is a maddening plethora of systems and units of measurement. To some extent I can blame the problems on the complexity of handling a constant space that must be addressed in several different ways and can also scroll across the screen. When player-missile graphics, with their independent coordinate system, are thrown in, the situation gets messier.

The first coordinate system keeps track of the cursor against the background of the map. This coordinate system is measured in units of color clocks and pairs of scan lines. Its basic unit is the smallest visual increment on the screen. This coordinate system sees the map as a gridwork 304 pixels high and 360 pixels wide. The position of the cursor in this system is recorded in zero-page addresses CURSXL, CURSXH, CURSYL, and CURSYH. This system is used for managing the scrolling functions.

The second coordinate system is a character-level version of the first system. This system measures the map as a gridwork 38 characters high and 45 characters wide. This system is useful for ascertaining the unit or terrain that the cursor is over. It is maintained with the zero-page variables CHUNKX and CHUNKY.

The third coordinate system maintains player-missile screen coordinates. It uses SHPOSP0 (shadow of horizontal position of player 0) and SCY (shadow of cursor Y-coordinate). This coordinate system is critical for all player-missile manipulations, for it is the only link between the scrolling map and the player coordinates.

The fourth and final coordinate system identifies the position of the map relative to the screen. It is useful for calculations involving the relationship between the map as a whole and the subset that the user sees. It uses the variables XPOSL, YPOSL, and YPOSH.

DATABASE

There are three primary database regions used by the interrupt service routines. The first is the data area on page zero in locations $B0-$BF. I allocated a good portion of my available page zero space for the interrupt routines because they are so time-critical. Most of the values stored here are coordinates. The second database region is the variable storage area on page six. This is used for single-byte variables (not tables) that have lower priority. Most of these values are also coordinates for the various graphics critters that run around on the screen. There are also a variety of counters and miscellaneous variables. The third database area for these routines is the database established by the data module. This consists of tables.

PERSONAL PROGRAMMING STYLE AND CONVENTIONS

A word on my personal programming practices is in order. Every programmer has little conventions about writing code and assigning labels. My conventions are simple. Labelled points that are merely the destinations of branches that skip over code are given meaningless labels. These points are typically not significant entry or exit points, but rather simple highway markers. I have found that trying to cook up descriptive labels for every destination point taxed my limited creative powers too heavily. I therefore adopted the simple expedient of labelling them in sequence X1, X2, X3, etc. up to X99. When I ran out of X's I went to Y, then Z, then A. This does not mean that I used 400 such labels. I wrote many sections of code that I later discarded; I discarded old labels along with old code.

Looping points were always assigned the label LOOPXX, where XX is a two-digit number. When I reached LOOP99, I went to LOOPA, LOOPB, etc. Only major entry points or truly significant program points received meaningful labels.

Variables are usually assigned meaningful names, although sometimes the references are obscure. I prefer to use defining suffixes rather than prefixes. Thus, coordinates will have an X or a Y suffixed to indicate their dimension. The suffixes LO and HI indicate the low order and high order bytes of a 16 bit number such as an address. CNT or NDX normally indicate some type of counter or index. FLG indicates a flag which is set to indicate a condition being met and cleared to indicate the condition not being met.

I always set aside several temporary variables called TEMPthis or TEMPthat. My rule for such variables is absolute: such a variable is always usable for very short-term storage and may never be used for storage exceeding one-half page of source code. I have a short memory.

VERTICAL BLANK INTERRUPT CODE

The VBI routine begins at $7400. It begins with a now-defunct break routine that I used for debugging purposes. This is a valuable tool for any serious programmer. It is prudent to build diagnostic tools into the software to facilitate debugging. This tool is keyed to the joystick in controller jack #2. You can jump out of the program and back into the Assembler/Editor cartridge by plugging a joystick into controller jack #2 and pressing the trigger button. I masked out the code by brute force in the final version. I believe so strongly in the importance of good debugging tools that I did not mask out the routine until the very last minute.

The next section of code handles the handicap option. It reads the console to see if the OPTION key is pressed. If so, it sets the handicap flag and changes the color in the text window. The change is effected by a rather sorry example of self-modifying code. I'm getting finicky about self-modifying code. To be worthwhile it really should do something surprising. This particular application accomplished nothing more than to save me a few minutes of programmer time and destroy any last shreds of respectability the program may have had.

The code beginning at line 2000 determines the state of the button and responds to it. It is tricked by the variable BUTMSK, a button mask set or cleared by the mainline routine to prevent the vertical blank interrupt routine from responding to the button. There are actually two conditions that must be tested. The first condition is the current state of the button, and the second is the state of the button in the immediately preceding VBI. The previous state of the button is recorded in BUTFLG. If both are false (neither the button is down nor was it down earlier) then we immediately proceed to test the joystick. Recall that the button-down condition is signalled by the critical bit being zero. If the button was down but isn't now down, then it was just released, and we must clear the text window and clear any flags and sounds that had been set. We must also unswap any unit in the cursor (more on this later). Finally, we clear out the maltakreuze and the arrow in case they were being displayed.

If the button was down and is still down, (BUTHLD) we must test the joystick for orders. First we check for a space bar being pressed; this would cause the orders to be cleared. Then we move the arrow (lines 2660-3330) until it reaches the maltakreuze. The task of moving the arrow is involved. The unit's orders must be retrieved and the relevant order must be stripped out of the byte. The arrow must be positioned and moved according to the order stored. Furthermore, the display is not done in a single pass of the vertical blank interrupt but in several. The speed of the arrow is set with the operand of the instruction in line 2630. The display of the maltakreuze is a somewhat simpler task (lines 3370-3590). The critical values for this routine are (BASEX, BASEY) which give the player-missile coordinates of the displayed unit, and (STEPX, STEPY) which give the player-missile coordinates of the arrow along its path.

The next button response routine is called FBUTPS and is the response to the first pushing of the button. This one does a lot of work. First it calculates (CHUNKX, CHUNKY) from the cursor coordinates. Then it attempts to find the unit (if any) underneath the cursor. This search alone can consume

1700 machine cycles. If it fails to find a match, the routine terminates. If it finds a unit under the cursor, then it must display the information on the unit.

The display routine is long (lines 4430-5350) but straightforward. The Y-register acts as an index into the text window for all display computations. As the characters are put into the text window, Y is incremented. The important coordinates BASEX, BASEY are computed in lines 5070-5240. These coordinates are exressed in the player-missile coordinate system. They are computed from the cursor coordinates SHPOS0 and SCY. Unlike the cursor, which can straddle map gridlines, they must be properly registered in the map gridwork. The computations in these lines center BASEX, BASEY on the unit.

The HMORDS and WHORDS values are shadowed out of their tables and into special locations on page six (HOWMNY and ORD1, ORD2). This is done in lines 5280-5340; its purpose is to make the orders processing simpler.

The orders input routine follows (lines 5390-6570). It is only entered when the button is held down and has been held down for at least one previous VBI. There are several error conditions which are tested before orders are entered (lines 5410-5660). These include giving orders to Russian units, exceeding eight orders, failure to wait for the maltakreuze, and entering diagonal orders. All errors result in a jump to SQUAWK, the nasty noisemaker routine which displays an error message.

This code also includes a debounce test. Simple switches bounce when first opened or closed, generating a sequence of on-off pulses spanning several milliseconds. A sufficiently rapid polling routine would read this sequence as many switch presses, and would enter multiple presses where the user had only pressed once. A common solution is to set a debounce timer that delays response to the entry for a period of time exceeding the bounce time. Such debouncing is automatically provided by the VBI routine's 16 millisecond polling period, but I inserted a very long debounce (160 milliseconds) anyway. I did this partly out of conservatism (never trust the machine to work properly) and partly to provide some protection against minor mistakes with the joystick. The delay of 1/6th second is not readily noticeable and gives some extra protection against errors.

The next chunk of code (lines 5700-5750) generate a feedback beep in response to the order. Next the new order must be folded into the existing orders. The task is to insert the two-bit order code specified by the joystick into the current orders byte. This requires some bit-twiddling. First we determine which of four bit pairs in the byte to use; the bit pair number is put into the Y-register and saved in TEMP1 (lines 5810-5870). Next we determine which of the two orders bytes should be twiddled. This byte index is either a 1 or a 0 and is put into the X-register (lines 5880-5930). Next, we shift the joystick entry bit pair upward in the byte to correspond to its desired position in the orders byte (lines 5940-6000). Lastly we fold our new order into the orders byte with a fiendishly clever bit of code that I learned from the fellows at Coin-Op (lines 6010-6050). Thanks, Mike and Ed.

The next routine repositions the maltakreuze (lines 6140-6360). This routine is a trivial memory move which moves bytes from the bit map table into the player RAM. It is of little interest.

The scrolling routine comes next. This routine is an adaptation of the routine I first distributed as SCRL19.ASM. If you are interested in the scrolling function of the game, I suggest that you purchase the Graphics/Sound Demo diskette containing SCRL19.ASM from the Atari Program Exchange, for it presents a far more general and better commented program for scrolling than this one. This scrolling routine differs from SCRL19.ASM in several ways. First, scrolling does not occur until the cursor bumps into an invisible wall near the edge of the screen. This is accomplished with some rather simple ad hoc tests in lines 7110, 7440, 7740, and 8220. The values tested were derived by trial and error. Second, the cursor motion is not uniform; it accelerates in the first second of motion. The purpose of the acceleration is to allow fine positioning without sacrificing speed. The acceleration feature is achieved with a very simple bit of code using variables called TIMSCL (time to scroll) and DELAY (delay between scrolls). By comparing TIMSCL with RTCLKL (real-time clock, low byte), the routine can determine when to move the cursor.

Fine scrolling is implemented by storing numbers directly into the fine scrolling registers. Coarse scrolling is implemented by accumulating a value called (OFFLO, OFFHI) and adding it to the LMS operands in the display list. This is done in lines 8650-8770. The final operation of the VBI routine is the preparation for the DLI routine. More on this later.

TABLES AND SUBROUTINES

The table JSTP is used by the artificial intelligence routine. DEFNC is used by the combat routine to figure the defensive value of a terrain type. DWORDS displays a fixed text message pointed to by an index in the accumulator.

SWITCH is an important subroutine. Its inputs are the coordinates of a square CHUNKX, CHUNKY and the identity of a unit CORPS. The subroutine then looks up the character code in the map and switches it with the value stored in the buffer table SWAP. This switches the unit character with a terrain character. The subroutine is used to bring units onto the map. At the beginning of the game there are no units on the map. Each one is brought in by subroutine SWITCH. Whenever the button is pressed and a unit is picked up, the subroutine is called to replace the unit with the terrain character. When the button is released, SWITCH is called again to put the unit back. SWITCH is also used to move units; they are switched off the map, their coordinates are changed, and they are switched back onto the map. SWITCH does not distinguish whether it is placing or removing a unit. A single call switches the unit character with the contents of its SWAP buffer; two calls in a row switch it twice.

The internal operation of SWITCH is simple enough. It computes an

indirect pointer (MAPLO, MAPHI) that points to the beginning of the map row containing the square. The Y-register provides the index to select the proper map byte. The computation of MAPLO, MAPHI is made simple by the fact that there are 48 bytes per map row. Multiplication by 48 is easy: four left shifts, a store, another left shift, and an add.

Subroutines CLRP1, CLRP2, and ERRCLR (lines 9900-10310) are uninteresting routines which merely clear out a player or an error condition and the text window. Nothing very fancy. BITTAB is used to select pairs of bits in a byte. ROTARR is a table used by the artificial intelligence routines to rotate an array. OBJX is a data table used by the artificial intelligence routine.

DISPLAY LIST INTERRUPT SERVICE ROUTINES

The display list interrupt routines are in lines 10450-11340. They are short, but very important. They are a curious mixture of cleverness and stupidity. The stupidity lies in the bucket brigade structure of the DLI execution. There are seven different DLIs serviced by this routine; the proper way to handle this many DLIs is to have each DLI rewrite the DLI vector to point to the next DLI service routine. The technique is described in Section 5 of DE RE ATARI. Instead, I used a DLI counter which is tested, bucket brigade fashion, until control finally reaches the proper DLI service routine. The time wasted by the technique is shameful.

The clever aspect of the code is the way that a DLI is applied to the map, even though the map is scrolled through the screen area. There are two character sets for the map. The switch from the northern character set to the southern one is made at CHUNKY=15. Unfortunately, we cannot simply set a DLI to hit on a specific mode line of the display, for there is no way of knowing if the map will be lined up with the screen properly. Indeed, with vertical scrolling taking place, the point where the transition should take place can be above, below, or on the screen. Obviously some cleverness is required.

The solution I used was to calculate during vertical blank the mode line on which the transition should take place. This value is calculated in lines 8790-8990 and is called CNT1. DLIs are set to hit on each and every mode line in the scrolling window. The DLI code will not be executed until the value of CNT1 indicates that the proper time has arrived. An alternative solution would have been to rewrite the display list every time a scroll is executed. There would then be at most one DLI bit set in the map window. The technique would have saved a great deal of execution time, and so it was the first technique I considered. As it happened, I encountered some difficult problems making the code work properly, so I gave it up and went to the present scheme using multiple DLI's only one of which does the work. The former method should be practicable; I don't know why I couldn't get it working. There's a lesson here: don't hold out for the elegant solution which eludes your grasp when an inelegant but workable solution is accessible. Readers of this document, a few score strong, will know what a klutz I am, but the thousands of happy users are none the wiser.

Another clever trick about these routines is in the timing. You may notice that they do not appear to be in a logical order. They have been carefully ordered to ensure that the most time-critical routines are at the front of the bucket-brigade, and the less critical routines are at the back of the bucket brigade. There is also a careful distribution of labor in the DLIs. Some graphics changes are made several lines before their effects are visible on the screen. This is one way of dealing with the shortage of execution time during a DLI. I make full use of the blank scan lines to perform some DLI chores. Blank lines are ideal for DLI's because no ANTIC DMA occurs during a blank line display; this leaves a full 55 machine cycles for Phase One DLI execution.

Finally, there is a strange example of tight timing in the last service routine. This routine is reached so late that it has almost no time before horizontal blank. I found that the STA WSYNC instruction sometimes produced skipped lines. This indicates that the instruction was being executed just as horizontal blank occurred. Rather than try to force horizontal blank synchrony, I decided to wait it out with a few time-killing instructions. It works.

On page 59 is a diagram depicting the sequence of changes made by the display list interrupts.

FINAL SUBROUTINES AND TABLES

Subroutine DNUMBR (lines 11390-11590) displays a number. It uses the table-driven method described in the notes on the data module. You can see that the code is certainly very clean and fast. Note that I was too lazy to properly encode the screen values properly, so I must perform a CLC/ADC #$10 which should have been done in the data itself. This waste of time in a very time-critical routine is not very consistent with my motivations which led to the use of this method.

NDX is a table used by the artificial intelligence routines to access bytes in an array.

XINC and YINC are tables used for motion. They tell how much to add to the X- or Y-coordinate given a step in any of four directions. I pulled an interesting stunt with YINC that shows how desperate I became for space. YINC is really a table 4 bytes long. The last 3 bytes of YINC just happen to be identical to the first 3 bytes of XINC. So I simply put the two together and cut out three bytes. This is a very dangerous way to save three bytes. If for some reason the two are separated, the program will malfunction in ways almost impossible to debug. Somewhere in the innards of my computer is an ugly green bug chuckling to himself. Someday he'll get me with that one.

OFFNC is a table of values used by the combat routines to evaluate attacks.

MAINLINE MODULE

This module handles the initialization of the game and game turn logic. It brings in reinforcements, figures the dates, seasons, and movement. The combat and thinking modules are subroutines called by this module.

I went through the module stripping out unnecessary equates to make the module somewhat smaller. This was necessary to make all of the source code fit onto a single diskette. You may wonder why I had so many unneccesary equates in the module in the first place. The five modules in this program must communicate with each other, and they do so through the variables in the database. This is impossible if the variables have not been declared in one of the modules. Furthermore, you can waste a great deal of time on bad assemblies discovering that some critical variable has not been declared. The ATARI Assembler/Editor cartridge is slow, and the printer slows things down even more. I solved this problem with a simple scheme. I wrote the modules in sequence. First the data and interrupt modules, then the mainline module, then the combat, and finally the thinking module. Each time I started a new module I created it by taking the previous module and stripping away all the code, leaving only the equates. This insured that each module inherited the complete database equate file.

There were two problems with this technique. First, I had to make certain that changes in an early file such as the interrupt module were properly transferred to all the succeeding modules. Also, equates in later modules sometimes needed to be included in the earlier ones. This problem plagued me throughout the development of the program.

The second problem with the all-inclusive database equate file is that the equate file eventually gets too large. The original equate file for the mainline module was four pages long. By stripping out some (but not all) of the unnecessary equates I was able to reduce it to only two and one-half pages. As you can see, there was a lot of fat. So if you see unused equates in the module equate files, don't get excited.

INITIALIZATION

The mainline routine begins with the initialization routines. The beginning of the mainline routine ($6E00) is the address to which the machine jumps after the program is fully loaded. The mainline routine must first initialize all of the hardware registers and database values. The first segment of code shows a common way to handle initialization of database variables. A table of initial values is kept with the main program. These values are then moved into the database region at the outset of the program. There is one danger in this technique: if for some reason you come along later and rearrange any of the database variables, the initialization code will put the numbers into the wrong places. This code forces you to keep all of your variables that require initialization together. It's not a bad idea to keep all such variables together, but it can be painful when you forget and make changes.

There will always be miscellaneous initializations necessary; with these you have no choice but to write a long string of LDA this, STA there,

instructions. The code is simple but you can waste a lot of bytes this way. One trick for reducing the size of this type of code is to group common initial values together. This is done in lines 1410-1460. Five very different locations all needed to be initialized to zero. Load once and then store five times. A similar method is used for several tables in lines 1480-1570.

The initializations in lines 1620-2060 are all quite straightforward.

MAIN GAME TURN LOOP

The outermost program loop begins on line 2080. The variable TURN is a simple turn counter telling which turn we are on.

First come the calendar calculations. These are simple enough. I add seven to the day, compare with the length of the month to see if a new month has arrived, and correct if it has. There is even a provision for the leap year in 1944 provided in lines 2190-2250. (At the time I wrote this routine I was planning to have the game cover the entire campaign.) With just a little effort the routine could be generalized to handle any leap year.

The tree color trick is executed in lines 2340-2350. Only two lines of code (6 bytes) and 13 bytes of table are required to implement the trick. Color register indirection can be powerful indeed, no?

Lines 2370-2670 put the date information onto the screen. They are simple data move routines with no interesting techniques.

The code in lines 2710-3080 is certainly the most obscure and clumsy code I have written in a long time. The purpose of the code is to figure out what season is in effect and perform any necessary changes related to the season. Unfortunately, I did not take the time to think the problem through. Instead, I just bulled into it, making up code on the fly and patching it together until it worked. The result is a gory mess.

There are four different variables (SEASN1, SEASN2, SEASN3, and EARTH) to tell the state of the season. SEASN1 is used to set the color of rivers and swamps. It holds a $40 for unfrozen water and a $80 for frozen water. SEASN2 tells if we are in fall or spring. This indicates whether the ice-line should move to the south or to the north. It holds a $00 to indicate spring and a $FF to indicate fall. SEASN3 is logically identical to SEASN2 but contains a different value becuase it is used in a different way. It holds a $01 in spring and a $FF in fall. EARTH is the color of the ground, brown for summer, grey for mud, and white for winter.

The code in lines 3130-3700 freezes the rivers and swamps. The algorithm here is interesting and instructive. The critical variables are ICELAT and OLDLAT. ICELAT defines the ice-line, that is, the latitude north of which everything is frozen. OLDLAT is the last turn's value of ICELAT. Everything between the two must be frozen. During spring, everthing between the two must be thawed.

The routine begins by calculating the new value of ICELAT. Notice that there is a random element in the determination of ICELAT. This randomness is leavened by cutting down the size of the random number (AND #$07) and adding a constant (ADC #$07). The result is a number ranging between $07 and $0E.

The code now prepares for the main loop which begins at LOOP40. It initializes LAT and LONG, which are input parameters for subroutine TERR. Then the loop begins. There are actually two loops beginning at LOOP40. The fundamental function of the loop is to sweep through all the map squares in the zone between OLDLAT and ICELAT, checking if they contain water. If so, they are then frozen or thawed, depending on the season. A complication is introduced by the presence of military units. The program must pick up each unit and look underneath to see what terrain is there, modify the terrain if necessary, and put the unit back down. This is gonna get messy, so hang on.

We begin LOOP40 by JSRing to subroutine TERR, an important routine that tells what type of terrain is in a given square. We specify the square's coordinates in LONG and LAT, and it returns the contents of that square in the accumulator. We then examine the terrain type. If it is the wrong type of terrain (mountains, for instance), we skip ahead to NOTCH (as in "no toucha da moichendize, eh!"), which proceeds to the next square in the row. If the square is touchable, we freeze or thaw it with the single instruction ORA SEASN1. Actually, we had already thawed it with the AND #$3F instruction in line 3390; the ORA instruction will freeze or ignore the byte depending on the value of SEASN1. In line 3540 we store the results of our crime. MAPPTR just happens to point to the right place because it is set up by subroutine TERR. Convenient, no?

As I said before, NOTCH moves us on to the next square. This is done by the simple expedient of incrementing CHUNKX. Of course, we must test to see if we have run off the edge of the map. This is done in line 3580. If we have reached the west edge of the map, we must reset CHUNKX and LONG to point back to the east edge of the map. Then we must go one step to the north or south depending on the season. This is done by adding SEASN3, which is either +1 or -1, to the latitude LAT. If we have not reached the vertical edge of the ice region, we loop back to LOOP40; otherwise, we exit the routine.

This is a big, slow routine. You can tell how slow it is by watching the freezing process in the game. You can actually see the iceline moving southward in November. Note that the routine is general enough that it can operate through many different years.

The next routine (lines 3720-3960) brings in reinforcements---units that have not been on the map up to now. This would be a simple routine if it weren't for one small problem: what if the unit comes in on top of another unit? We can't have that, so before we place the unit we have to see if anybody else is already there. This is all done in lines 3760-3840. Lines 3850-3880 notify the player of the arrival of reinforcements. If a unit was not allowed entry onto the board, lines 3910-3940 make sure that he'll get another chance next turn by modifying his value of ARRIVE.

Logistics is handled in lines 3980-4030. It is a simple loop with a subroutine call. The subroutine is inside the combat module; it is discussed in the essay on that module.

POINTS CALCULATION

Lines 4070-4760 calculate the current point score of the player. The algorithm used is involved. There are three factors used in calculating points: 1) how many German muster strength points have been projected how far east, 2) how many Russian combat strength points have been projected how far west, and 3) how many special cities have been captured by the Germans. I feel that this routine is instructive as a good example of a fast, short, and simple routine that imposes reasonable and realistic demands on the player. The importance of the routine is the algorithm, not the coding. The algorithm is optimized for the strengths and weaknesses of the 8-bit processor. Let's look at the implementation closely.

The routine starts by zeroing ACCHI and ACCLO, as these together constitute the point counter, which is sixteen bits wide. It then enters a loop that calculates the points for moving German units east. The longitude of each German unit (CORPSX) is subtracted from a constant value of $30. This value is multiplied by MSTRNG/2 in lines 4190-4280. The multiplication is the stupidest kind: a simple repetitive addition. For single-byte quantities the technique is not too expensive in time. Unfortunately, I did not analyze the problem carefully and so I got the looping backwards. The value of MSTRNG/2 is the loop counter in Y and the value of $30-CORPSX is the added constant. The former value will almost always be larger than the latter, so I should have used the latter as the loop counter. It's always faster to add, say, 50 to itself 3 times than to add 3 to itself 50 times. Oops.

After German points are calculated I begin calculating the effect of Russian points. These will be subtracted from the points accumulated by the Germans in the first loop. For the Russian units, a slightly different algorithm is used. First, the combat strength, not the muster strength, is used. Why? I didn't want to penalize the Germans for moving to the east. Remember, during winter the Germans have a harder time getting supplies as they move further east. So I had to use their muster strength. I also wanted to reward the Germans for Russian units that were still on the board but out of supply. So I used combat strength for the Russians.

The sum of the Russian score is subtracted from the German score in lines 4550-4590. Lines 4600-4680 award point bonuses for capturing cities. A simple loop is used. Two tables drive this routine. One, MOSCOW, is a simple set of flags that tell if the cities have been captured. The other, MPTS, holds the point values for each of the cities. If MOSCOW is set, the number of points assigned for that city are added to the point score.

The final operation associated with point evaluation is to halve the total points if the handicap was used. The operation takes three lines

(4700-4720).

Once the points have been calculated, they must be displayed. This is done in lines 4730-4760 in an operation which by now should be familiar to the reader. Next comes a test for end of game. The termination is not particularly elegant. I simply put an endgame message onto the screen and hang the game up in a loop. I am sure a more elegant termination could have been arranged but I was too lazy to implement one.

Lines 4850-4930 deal with the artificial intelligence routine. They allow the player to use the joystick button (by clearing BUTMSK) and put a prompting message on the screen. Then they jump to the artificial intelligence routine. The program spends most of its time there. It does not return until the player presses the START button. Then the joystick button is masked out by setting BUTMSK and an appropriate message ("figuring move---no orders allowed") is put onto the screen.

MOVEMENT EXECUTION

Lines 4970-5030 prepare the way for movement execution. They initialize the subturn counter TICK and calculate the first execution time of each unit. As mentioned in the player's manual, each turn is broken into 32 subturns. The movement cost to enter a square is expressed in terms of the number of subturns necessary to wait before entering the square. Subroutine DINGO does this calculation. The name DINGO is absolutely meaningless. You should see some of the labels I have used in other programs. When I was an undergraduate doing physics programs I had a penchant for obscene labels. It made sessions with the consulting programmer (especially lady programmers) interesting. The only problem with the idea is that there are a limited number of four-letter words, and I was forced to recycle each word in many different incarnations. Later on I took to using names of animals, fruits, foods, anything. I can't stand acronymic gibberish. I prefer creative gibberish.

Lines 5050-6180 perform the movement. The outer loop beginning with LOOP33 sweeps through all of the subturns. The inner loop beginning with LOOP32 sweeps through all of the units. The inner loop begins by performing the combat strength recovery function. If the combat strength is less than the muster strength, it is incremented. If the difference between the two is large, the combat strength may be incremented again. This ensures that large units will recover combat strength faster than small units.

The most heavily used test is at lines 5180-5190. This determines if the execution time of the unit has arrived yet. If not, the loop proceeds to the next unit.

An interesting stunt is pulled here. Program flow goes through line 5500, which is merely a jump instruction. You may wonder why I didn't insert a jump at the original branch point. I did it to save a few bytes of memory. Any big loop will have a variety of tests that call for abortion of the main loop and immediate procession to the next iteration. If the loop is

considerably longer than 128 bytes, the 6502 branch instructions will not work. The standard response to this problem is to replace the branch with its logical inverse (e.g., BCS with BCC or BNE with BEQ) and follow it with a JMP instruction. This costs three extra bytes. The waste can be reduced by placing a JMP instruction halfway through the loop and having the local test points branch to it. It acts rather like a collecting station for loop abortions. Three bytes are saved for each abortion path.

The remainder of the movement code retrieves the unit's orders, examines the terrain in the destination square, and checks if it is occupied. If it is occupied by a friendly unit, the moving unit must wait two subturns (lines 5450-5490). If it is occupied by an enemy unit, combat occurs and is referred to the combat subroutine at $4ED8. If the unit is allowed to enter the square, either because it was victorious in combat or the square was unoccupied, the code at DOMOVE, lines 5550-6060, is executed.

One last test must be made before actual motion happens. Zones of control are tested in lines 5550-5740. If zones of control do not interfere, the unit is moved by SWITCHing it off the map, substituting the new coordinates as parameters for SWITCH, and SWITCHing it back onto the map. The now-executed order is deleted from the unit's orders queue (lines 5850-5920). A test is made to see if the unit has entered a victory city. If so, the flag for that city is set or cleared depending on the nationality of the moving unit. This is done in lines 5930-6060. Lastly, the execution time until the next order is calculated by DINGO. Then the loop goes to the next unit. When the last unit has had its chance to move, the subturn counter TICK is incremented; when TICK reaches 32 a new turn begins. With this the major loop terminates.

The remainder of the module is devoted to subroutines and tables. STALL is a delay loop that kills time to slow down the action during movement. The debugging routine that follows (lines 6420-6610) links with the debugging routine first mentioned in the interrupt discussion.

TERR is a major subroutine. It sets up a pointer to the map (MAPPTR) on page zero and retrieves the contents of the map at the coordinates LAT and LONG. If the square contains a military unit, it determines the identity of that unit as well as the terrain underneath the unit. Note that TERR returns a terrain code identifier in the accumulator and unit identity (UNITNO). It also returns the terrain identifying code in TRNTYP. It also returns the Z flag of the 6502 processor status register set if the square was indeed occupied. Many calls to TERR are immediately followed by a BEQ or BNE instruction; such calls are attempting to determine if a square is occupied.

TERR does contain an interesting tidbit. Lines 7220-7230 are strictly error flag lines. They put an asterisk onto the screen. If these lines are ever executed a program error has occurred. The error arises when TERR finds a unit character in a square but is unable to find a unit whose coordinates match those of the square. It turned out that this condition could arise from a large number of bugs created by other sections of code. I would never find out about the problem during testing until it was too late to track the bug down. So I put this code in to warn myself. As it happens, there is

another symptom of the bug that is more interesting. The program becomes confused and starts mixing terrain codes with unit codes. The next thing you know, trees, cities, and rivers are marching around the map, fighting battles, retreating, and carrying on in very unterrainlike ways. I tracked down this bug diligently; I believe that it is now quite dead.

Subroutine DINGO is the next subroutine in sequence. It looks up a unit's orders, finds out the terrain in the destination square, determines the delay imposed by that terrain, and stores the delay in the unit's EXEC storage.

Subroutine TERRTY determines the type of terrain in a square, given its character code (TRNCOD). There are several different character types for each terrain type, so some logical analysis of character types is necessary to determine terrain types. It is done with a simple bucket brigade of logical tests. Somehow I am sure that there is a neater way to do this.

ZPVAL is a table of initial values for page zero locations. PSXVAL is a similar table of initial values for page six locations. COLTAB is the table that specifies tree colors for each month of the year. MPTS gives the point scores allocated for each captured city. MOSCX and MOSCY give the coordinates of cities that earn points. TXTMSG is a very simple subroutine that puts a 32-byte text message onto the screen.

COMBAT MODULE

This module handles combat resolution and logistics for the mainline routines. It is nothing more than a set of subroutines called by the mainline routines as needed. Hence, its layout and structure are simple.

The fundamental design of the combat system is not obvious. All combat systems have as their inputs the strengths of the opposing units and the environmental conditions under which they fight. All such systems attempt to determine outcomes as functions of these input conditions. The normal outcomes are reductions in strength and retreats. This game has two types of strength to reduce, which adds some richness to the possibilities.

The unique aspect of this combat system lies in the iterative nature of the combat results system. Instead of trying to compute the outcome of the battle with a single formula, this routine breaks a week-long battle up into many tiny battles which are resolved by simple rules. Each mini-battle can kill only a small number of muster and combat strength points on each side. Thus it is the aggregate effect of many such battles that determines the overall outcome of the battle. The sensitivity and power of the combat results system arises from the statistical behavior of this ensemble of many small battles.

This raises a very important point in game or simulation design: many very advanced functions can be generated using iterative methods with very simple arithmetic. Many people claim that good simulations cannot be done on microcomputers because 8-bit arithmetic is not good enough. While it is certainly true that eight bits are hard to work with, we must remember that eight bits of resolution give better than one percent accuracy in stating a properly normalized number. With imaginative programming these machines can do a great deal of impressive simulation.

SOUND AND GRAPHICS EFFECTS

The module begins with the combat resolution routine at $4ED8. It first clears the flag VICTRY, which is used to tell the mainline routine if the attack was successful. If so, the attacking unit will be allowed to enter the square it attacked. It then checks the attacking unit (ARMY) to make sure that it is not a Finnish unit. Finnish units are not allowed to attack.

The next step (lines 1270-1400) is to create the combat graphic in which the defending unit flashes in solid color. This is done by replacing the unit's original representation on the map with a solid square of color. There must be some logic to determine the nationality of the defending unit (red for Russians, white for Germans). The character used is simply the solid character used for the borders of the map and the open seas. We'll replace the original character later on.

Now we must make the machine gun sound. This is done in lines 1410-1520. My original intention was to create a deep explosion sound, rather like artillery. The result was not at all what I expected, but I liked it so much I left it as it was. The loop in lines 1430-1520 changes the frequency and the volume of the sound produced. The sound is stretched

out with subroutine STALL from the mainline module.

A great deal of time is killed in this loop, deliberately so. When I first ran these routines with no delays the motion and combat happened so fast that I had no chance to observe what was happening. I pushed the START button and saw pieces flying all over the screen like banshees. It was all over in less than a second. I decided that the player would enjoy sweating his turn out, so I put in longer and longer delays until it seemed right.

In lines 1560-1590 I put the defending unit's piece back on the map. The rest of the routine will execute very quickly.

COMBAT RESOLUTION

In lines 1620-1760 I evaluate the factors affecting the defender's strength. There are three: the defender's combat strength (CSTRNG), the terrain that the defender lies in, and the motion of the defender. Terrain evaluation is simple. Terrain can halve, double, or not affect the defender's strength. Notice the test on lines 1690-1700. This protects against overflow. If a large number is doubled too much it can overflow and produce a small number---an unfortunate inaccuracy. I guard against this by monitoring the Carry bit and reloading an $FF if it strikes.

In lines 1740-1760 I implement a very simple rule: defenders who are moving at the time they are attacked have their defensive strength halved. The implementation is about as clean and simple as you can get. This makes an important point about designing with a microcomputer. Some things are trivially simple to do; this operation requires six bytes of code and nine cycles of execution time. Other operations, such as logistics evaluation, are painfully difficult to execute. A designer needs a feeling for what can be done easily and tries whenever possible to work with the grain of his machine rather than against it. Of course, if he/she is to produce anything interesting, he/she must eventually cut across the grain. Doing it well is the hallmark of brilliant design.

In lines 1800-1900 the defender gets to make a first strike against the attacker. The defender's adjusted combat strength in the accumulator is compared with a random number. If it is less than the random number, the defender's pre-emptive strike fails and the attacker makes his strike. If it is greater, the strike succeeds. The attacker suffers the standard loss: he loses one point of muster strength and five points of combat strength. A test is then made to see if the attacker dies or breaks. More on death and breakage later.

On line 1940 we begin the main point of the whole routine, indeed of the whole game. ("The decision by arms is for all operations in war what cash settlement is in trade"---Clausewitz). We figure the attack. The only adjustment made on the attacker's combat strength is the halving of attack strength if the attacker is on a river square. Then we compare the attacker's strength with a random number just as we did with the defender. If the attacker's adjusted combat strength is less than the random number,

the attack fails and the combat routine terminates. If it is greater, then the attack succeeds and many things must happen. First, the defender loses one muster strength point and five combat strength points. That's easy enough to execute (lines 2100-2140).

Next, we must check if the defender dies. If so, we jump to subroutine DEAD, which handles all the paperwork for killing units. This is surprisingly extensive. His combat strength, muster strength, and orders must be zeroed. His execution time and arrival times on the map must be set to nonsense values to preclude his reincarnation. Finally, the body must be removed from the map with subroutine SWITCH.

If the defender did not die, we then test for breakage. This is an important concept in the game. A unit will stand and fight up to a point. At some point morale will break and the unit will collapse and run. Research has shown that this most often happens when some fraction of the unit's strength is destroyed. I chose to measure the intensity of a unit's casualties by comparing the unit's combat strength with its muster strength. If the combat strength falls below some set fraction of the muster strength, the unit breaks. The fraction used depends on the nationality of the unit. German and Finnish units were fairly tough; they don't break until their combat strength falls below one-half of their muster strength. All other units break when their combat strength falls below seven-eighths of their muster strength. The calculations for this are carried out in subroutine BRKCHK, lines 4980-5200. Any unit that breaks forgets any orders that had been assigned to it. Your priorities change when you're on the run.

If the defender does not break, the combat routine terminates. If he does break, he must retreat. This is a complex procedure; it is executed in lines 2210-2750. The basic idea of this code is that the defender attempts to retreat in various directions, but can find his retreat path blocked by zones of control, enemy or friendly units, or open ocean. If any of these events occurs, the unit suffers a penalty and attempts another route. If no retreat path is available the unit suffers heavy losses and remains in place.

An important subroutine for this retreat process is RETRET (lines 2850-3410), which checks for the various conditions that block retreats and exacts the penalty for blocked retreat paths.

If the defender can retreat, the retreat is executed in lines 2500-2630. The victory flag is set to tell the mainline routine that the attacker may indeed move into the defender's square regardless of the presence of enemy zones of control.

The combat routine terminates by incrementing the execution time of the attacking unit.

LOGISTICS

The supply evaluation routine is the next major routine in the module. The basic idea of the routine is to start at the location of each unit and

trace a line from that unit to the appropriate edge of the map without encountering a blocking square. A blocking square is a square containing an enemy unit, a square in an enemy zone of control (unless occupied by a friendly unit), or an open sea square if the unit is Russian. If a blocking square is encountered, the routine must try to trace the line in another direction. It is very easy in such circumstances for a routine to hang up in an infinite loop bouncing between two blocked squares. I precluded this by the clumsy solution of counting the number of blocked squares encountered and declaring the line blocked when the count exceeded a critical value. This critical value depends on the nationality of the unit and the season. There are also seasonal effects on German units. During mud, they receive no supplies at all. During winter, the probability that a German unit will receive supplies depends on how far east the unit has gone. The further east, the smaller the probability. Let's see how all this is done.

The first thing to do is skip units which have not yet arrived on the map (lines 3450-3490). In line 3510 I determine the nationality of the unit. If it is Russian, I skip the weather determination section. Notice the redundant code on line 3530. I blew it. I determine the season in lines 3540-3550 by examining the color of the ground. That's the simplest way to find out the season. If it is mud, there is no supply, period. If it is winter, then I perform a rather odd calculation. I quadruple the unit's longitude and add $4A. This guarantees that the resulting number in the accumulator will lie between 74 and 254. This number becomes the probability (measured against 255) that the unit will receive supplies. Thus, Germans on the west edge of the map have about a 99 percent chance of getting supplies while Germans on the east edge of the map have only a 30 percent chance.

There are two major loops in the logistics routine. The inner loop, labelled LOOP90, attempts to choose a safe direction in which to move from the current square. The outer loop, LOOP91, performs the jump to the chosen square. The inner loop always attempts to jump towards the home map edge (HOMEDR). If that fails, it attempts random directions until it finds a way out or it runs out of tries.

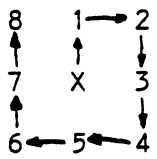
After supplies have been figured, any Russian units in supply have two points added to their muster strength. This is a Russian advantage.

ZONE OF CONTROL

The next routine tests for zones of control. Specifically, it answers the question, "Is there an enemy zone of control extending into square (LAT, LONG) for a German/Russian unit?" The algorithm used is as follows: Examine the square in question to see if it is occupied by an enemy unit. If so, the square is automatically considered in a zone of control. If it is occupied by a friendly unit other than the unit in question, then the square is automatically out of any zones of control. If the square is unoccupied, then we examine all surrounding squares to determine if they are occupied by enemy units. Units in corner squares add one to the ZOC counter. Units in directly adjacent squares add two to the ZOC counter. If the ZOC counter equals or exceeds two, a zone of control is cast into the square.

The routine begins by zeroing the ZOC counter. Then it sets the TEMPR register with a value that identifies the original unit's enemy as either Russian ($40) or German ($C0). Then it examines the contents of the square by calling TERRB. If the square is unoccupied, it branches ahead to A74. If it is occupied, it compares the nationality of the occupying unit (AND #$C0) with that of the original unit (CMP TEMPR). If they are equal, it is an enemy unit and the routine immediately sets the ZOC counter and terminates. If they are unequal, it is a friendly unit and the routine must find out if it is the same as the friendly unit. This is done by comparing coordinates (lines 4410-4460).

If the square is unoccupied, the surrounding squares are examined by a sneaky scheme. There is a table in memory called JSTP+16 that holds jump vectors for a walk around a square. The system works like this:



Starting at X, and proceeding in sequence around X as indicated by the numbers, the sequence of steps is:

(0=north 1=east 2=south 3=west)

0, 1, 2, 2, 3, 3, 0, 0

These are the values seen in the JSTP+16 table, backwards for the 6502's countdown capability. Thus, to execute a walk around the square X, we execute jumps in the directions specified in the JSTP+16 table. The complete walk around the square is executed in lines 4510-4740.

THE IMPORTANCE OF ALGORITHMS

This routine demonstrates a very important principle of software design: the best way to improve performance is to re-examine your algorithms very closely. When I first wrote this routine it was very large and slow. The original algorithm was simple and obvious, but much too slow. It examined each and every unit in turn, subtracting its coordinates from those of the square in question. If the difference of both sets of coordinates was one, the two units were diagonal to each other and I incremented the ZOC counter. If the difference of one pair of coordinates was zero and the other difference was one, then I added two to the ZOC counter. The algorithm is fairly obvious but it required over 200 bytes of code and a very long time to

execute. I tried many of the standard means of speeding it up, but they made it even bigger. I finally grew desperate enough to carefully rethink the entire algorithm. After much brainstorming I came up with the current algorithm, which is subtler but much more efficient. I saved nearly a hundred bytes of code and cut the execution time for typical operations to a third of its previous value. The moral of the story is, rethinking your algorithms will frequently net you far more performance than any amount of clever coding.

THINKING MODULE

This module handles but one task: the artificial intelligence for the Russian player. It has one entry point at $4700 and one exit point at $4C22. It includes several subroutines and data tables for its own use. Thus, this is the most direct and straightforward routine of the entire program. Unfortunately, it is also the most involved routine of the program. It is also the biggest, including about 1.5K of code. To make matters worse, it is almost devoid of comments. This module was one of the best-planned modules in the entire program. For this reason I felt little need to comment on it as I was writing the code. That just makes the task more difficult now.

The basic goal of this routine is to plan the moves of the units. This translates into the specific task of producing values of WHORDS and HMORDS for each Russian unit. Many factors must be considered in computing the orders for each unit. The routine must determine the overall strategic situation as well as the local situation that the unit finds itself in. This will tell whether the unit should think in terms of attack or defense. The overall situation is determined by computing the danger vector. The danger vector tells how much danger is coming from each of the four directions.

The unit must evaluate the four possible directions it can move in. Each direction must be evaluated in terms of the danger vector, the nature of the terrain, the impact of the move on the integrity of the Russian line, the possibility of traffic jams, and the presence of German units. All of the surrounding squares must be evaluated and the best one chosen.

The really difficult aspect of the decision-making process is the necessity of coordinating the moves of all Russian units. The problem is made vastly more difficult by the fact that we must coordinate each unit's possible move with the possible moves of all the other units. The possibilities multiply in a truly mind-boggling manner. My solution was rather esoteric. Imagine the Russian army lying in its positions at the beginning of a turn. Imagine now a ghost army of virtual Russian units, initially springing from the real army, but with each ghost army plotting a path of its own across the map. Each ghost plans its path based on the assumption that the other ghost armies represent the concrete reality that must be conformed to. Thus, each ghost in turn says, "Well, if you guys are gonna move there, I'm gonna move here." One at a time, the ghost army adjusts itself into new positions. This process can continue until each ghost can say, "If you guys are gonna be there, I'm gonna stay right where I am." In practice this situation is almost achieved after only about ten iterations. However, if the player presses the START button, the iterations stop and the ghost army becomes the destinations for the real army. In this way hypothesis is converted into plans.

OVERALL FORCE RATIO

The module begins at line 1680. The first task is to calculate the overall force ratio. This is the ratio of total German strength to total Russian strength, and is a useful indicator of the overall strategic situation. To calculate this number, we must first add up the total German strength and the total Russian strength. This calculation is made in lines

1730-1870. The upper byte of the total strengths is stored in TOTGS (total German strength) and TOTRS (total Russian strength).

The next problem is to calculate the ratio of these two numbers. This is a simple long division. Unfortunately, I was not prepared to do a long division. Such arithmetic takes many machine cycles to crunch and many bytes of code to do properly. The floating point arithmetic package provided in the Operating System ROM did not interest me. So I wrote my own special routine to handle the problem. This is an example of individual crotchetiness, not judicious planning. I probably should have used the floating point package, or at least a decent 16-bit integer arithmetic package, but I was too lazy and impatient.

The first problem I must solve arises from the high probability that the total German strength is going to be very close to the total Russian strength. If I take a straight ratio of the two I will very probably get a result of 1. Since I will have integer arithmetic, my result won't be very sensitive to changes in the total strengths. I solved this problem by arbitrarily multiplying the ratio by 16. It's my program and I can cheat on the arithmetic if I want to.

Unfortunately, multiplying by 16 creates a new problem. Should I multiply the quotient by 16 or divide the divisor by 16? Either approach will have the same effect, and both approaches have the simplicity of being executed with simple logical shifts. But dividing by 16 loses some precision in the quotient, and multiplying by 16 runs the risk of losing the whole number. For example, what if total German strength is 17 and I multiply by 16 by ASLing four times? I don't get 272 for an answer, I get 16. Check it out for yourself.

Here's the clunky solution I came up with: ASL the dividend (line 1950) until a bit falls off the high end of the byte into the Carry bit (line 1960). Put it back where it belongs (line 1970) and then LSR the divisor (line 1980) the remaining number of shifts.

Now I am prepared to do a dumb long division (lines 2070-2140). Load the dividend into the accumulator. Keep subtracting the divisor from it until it is all gone. The number of times you subtract the divisor is the quotient. It's dumb, it's slow, but it works. More important, I can understand it. The final result is stored in OFR, the overall force ratio.

INDIVIDUAL FORCE RATIOS

The next task is to calculate the individual force ratios. The war might be going really well for Mother Russia, but the 44th Infantry Army may not find conditions as rosy if it is surrounded, out of supply, and being attacked by four Panzer Corps. It is necessary to supplement global planning with a local assessment of the situation. This is expressed in the individual force ratios. There are five individual force ratios: Four express the amount of German danger bearing down on a Russian army from the four cardinal directions. The fifth expresses the average of these four.

The fifth is called the individual force ratio (IFR). The other four are called the IFRN, IFRS, IFRE, and IFRW, for the directions they represent.

SUBROUTINE CALIFR

Subroutine CALIFR (lines 8390-9690) calculates the individual force ratios. This is an extensive computation which requires a great deal of time and memory. The fundamental idea behind this subroutine is that danger is a vector, having both a magnitude and a direction. This subroutine determines aggregate magnitude and the aggregate sum of the danger to the unit.

The subroutine begins by zeroing the local variables IFR0, IFR1, IFR2, IFR3, and IFRHI. These correspond to the IFRN, IFRE, IFRS, IFRW, and IFR tables, but are easier to use in the routine. After initializing some coordinate variables, the first large loop begins.

This loop, beginning with line 8520, extends all the way to line 9230. Its purpose is to calculate the directional IFRs, so it is really the meat of the subroutine. It sweeps through each unit, first checking if the unit is on the map (lines 8520-8540). If so, it determines the separation between the tested unit and the unit whose IFR is being computed. It measures this in terms of both the total distance between the two (ignoring Pythagoras) and the X-separation (TEMPX) and the Y-separation (TEMPY). Units further than eight squares away are considered to be too far to be of any local consequence (lines 8680-8690). The range to closer units is halved and stored in TEMPR.

The unit's combat strength determines the magnitude of the unit's threat. We must also calculate the direction to the unit. This is done in lines 8750-9020. These lines test the direction vectors to determine the overall direction to the unit. The result of these tests is a value in X of 0, 1, 2, or 3. This value specifies the direction of the threat.

In lines 9030-9150 we determine the magnitude of the threat. We get the combat strength of the tested unit, divide by 16, and check to see if the tested unit is Russian or German. If Russian, the result is added to the running sum of local Russian strength (RFR). If German, it is added to the running sum of local German strength in the direction specified in the X register. This done, program flow loops back to the next unit in sequence.

The next chunk of code, lines 9250-9320, add up all the danger values from all four directions and leave the result in the accumulator.

The next chunk of code, lines 9350-9570, calculates the final individual force ratio in much the same manner that the overall force ratio was calculated. The dividend is multiplied by 16 (lines 9350-9420), and then the divisor is subtracted from the dividend repeatedly until the dividend is all gone (lines 9450-9510). The count of the number of subtractions equals the quotient. This quotient is averaged with the overall force ratio (lines 9540-9560) and the result is stored in the IFR for the unit. The only remaining function is to move the local directional IFRs to the unit-specific

IFR tables (lines 9610-9680).

Subroutine INVERT is a simple absolute value routine. It takes a value in the accumulator and returns the absolute value of the number in the accumulator. You may have noticed that it was used heavily in the code. By JSRing to INVERT+2, we get the negative value of the accumulator returned.

Back in the main part of the module, we complete the IFR loop by setting the army's current position (CORPSX, CORPSY) to the objective position (OBJX, OBJY). OBJX and OBJY are the coordinates of our ghost armies. This completes the initialization loop. We now enter the main loop of the program.

MAIN LOOP STRUCTURE

The main program loop begins on line 2340 and extends all the way to line 7290. It is obviously a gigantic loop, and it takes a long time to execute. It is also an indefinitely terminated loop. It does not terminate after a specific number of passes. It keeps looping until the player presses the START key. The main loop sweeps over the entire Russian army. The inner loop sweeps over each unit in the Russian army.

The first task of the loop is to ignore militia armies and armies that are not on the map. Militia are not allowed to move. If an army does not fail these two tests in lines 2360-2420, then the local military situation for the army is evaluated. This is done by comparing the army's individual force ratio with the overall force ratio. If IFR=OFR/2, then the army must be more than eight squares from the nearest German unit. This conclusion can be made from the way that CALIFR calculates the IFR. If the army is far from the front, then it is treated as a reinforcement. If not, it is treated as a front-line unit, and a different strategy is used.

REINFORCEMENT STRATEGY

The job of a reinforcement is to plug weak spots in the line. This requires that the unit be able to figure out where the line is weak, no easy task. The trick is to use the existing Russian front-line units as gauges for the seriousness of the situation at any segment of the front. Where the front is solid, the IFRs of the front-line units will be low. Where the front is weak, their IFRs will be large. So we need merely examine the IFRs of all Russian units, select the largest, and head in that army's direction. Well, not quite. We don't want all the reinforcements heading for the same spot or the beleaguered Russian army will find himself trampled by his rescuers. More important, we need to take into account the distance between unit in distress and rescuer. There is no point in rushing to save somebody several thousand miles away.

The code to do all this extends from line 2470 to line 2870. The section starts by initializing BVAL to the value of OFR/2. BVAL stands for "best value" and is used to store the value for the most beleaguered Russian

army. Then a loop begins at line 2520 which sweeps through all Russian armies, rejecting off-map armies and calculating the separation between the tested army and the reinforcing army. This separation is divided by 8 (lines 2660-2680). I cannot now figure out the purpose of the branch in line 2690. It throws out the tested army if the separation had bit D3 set. A very strange test indeed. Lines 2700-2760 subtract the separation from the tested unit's IFR and compare the result with the best previous result. If the new result is bigger, then this unit has a better combination of proximity and (get this) beleagueredness. This unit becomes the preferred unit. Its value is stored in BVAL and its ID number is stored in BONE (best one). Then we move on to test another unit. When all units have been tested the best one is selected for support. Its coordinates become the objective of the reinforcing army. The job of planning that army's move is done and the routine jumps to the end of the loop (TOGSCN).

STRATEGY FOR FRONT-LINE ARMIES

Front-line armies have a very complex strategy. They must evaluate a large number of factors to determine the best possible objective square. These factors are: the army's IFR, its supply situation, the accessibility of the square, the straightness of the line that would result, the vulnerability to being surrounded, the danger imposed by nearby Germans, the possibility of a traffic jam, the terrain in the square, and the distance to it. Let's take it slowly.

In lines 2990-3050 we perform a simple test to see if the unit should take emergency measures. We ask, is the army seriously outnumbered? Is it out of supply? If either answer is yes, then this army is probably trapped behind German lines and it must escape to the east. It is given an objective square directly east of its current position. It will frantically crash eastward, regardless of the circumstances. It will even attack vastly superior German units in its haste.

This may strike you as pretty stupid. I gave a good deal of thought to the problem and I am convinced that this is the best all-round solution. My first solution was much more intelligent: I had such Russian units run away from the Germans. This normally meant that they ran to the west, straight for Germany. This is not very realistic. It also forced the player to assign large numbers of troops the boring job of tracking down and finishing off the forlorn Russian armies. I considered having cut-off Russians sit down and stay put, but then they would never have any chance of escaping. Quite a few Russians do indeed escape with this system, so I think it has proven to be a successful way of dealing with a difficult problem.

NORMAL FRONT-LINE ARMIES

If an army is not in trouble then it must choose a direction in which to move. The computations for this choice begin in line 3130, with DRLOOP, the direction loop. The critical loop variable is DIR, the direction of movement being evaluated. For the purposes of this loop, DIR takes the following

meanings: 0=north, 1=east, 2=south, 3=west, FF=stay put. This loop answers the question, "Should this army move in direction DIR?" It first determines the square being moved into (lines 3160-3240). The coordinates of this target square are TARGX, TARGY. The square being left is a ghost army square at OBJX, OBJY. The value of this target square is SQVAL. After verifying that the square can be entered (lines 3290-3340), the primary logic begins.

LINE INTEGRITY COMPUTATIONS

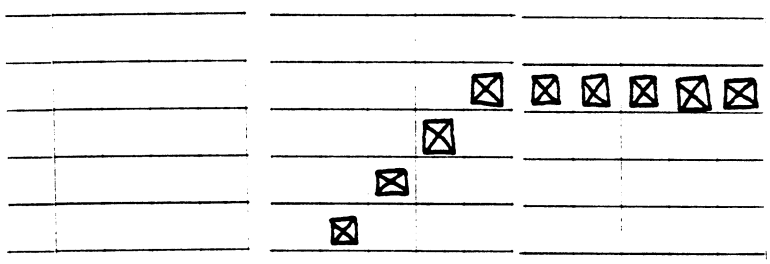
To figure whether a move will result in a solid line or a weak line, it is first necessary to give the computer some image of what that line looks like. I did this by creating two arrays. The first array is called the direct line array and is stored in LINARR. This array is 25 bytes long and covers a 5-by-5 square. The square being tested is always at the center of the big square. The routine will not evaluate the entire Russian line, for that task is impossibly large. Instead, it will treat it as a collection of short line segments and evaluate each segment for desirable configuration.

The big square is addressed by starting at the central square, whose coordinates are TARGX, TARGY, and stepping outward in a spiral from this square. The direction vectors for this spiral path are specified in a table called JSTP. The counter for the steps is called JCNT. The coordinate of a little square being considered within the big square is always SQX, SQY.

The contents of the big square are computed with two nested loops, LOOP56 and LOOP55 (lines 3450-3800). The outer loop steps through each of the 25 squares in the big square (except the central square, which we assume will contain the ghost army). The inner loops sweeps through all Russian armies to see if one's objective is in the square being tested. Note that we check not for the presence of the unit itself (CORPSX, CORPSY) but rather for the intention of the unit to go to the square (OBJX, OBJY). This is how we coordinate the plans of the different armies. If a match is obtained, the muster strength of that army is stored into the array element (lines 3760-3780). We then store the muster strength of the army whose plans are being made into the array element for the central square. When this task is completed we have an array, LINARR, which tells us how much Russian muster strength is in each of the 25 squares surrounding the square in question. We can now examine the structure of this configuration. We will examine it from four different directions: north, south, east, and west. We will keep track of which direction we are looking from with the variable SECDIR (secondary direction).

THE LINE VALUE ARRAY

A very useful tool for examining this two-dimensional array is to construct a one-dimensional representation of its most important feature. This one-dimensional representation will answer the question, "How far forward is the enemy in each column?" A picture might help:



LV ARRAY: 5, 5, 5, 5, 5 5, 4, 3, 2, 1 1, 1, 1, 1, 1

If a particular column is not populated at all, the value in the corresponding LV entry is five.

Lines 3920-4220 build the LV array from the LINARR. The variable POTATO (remember I told you I sometimes used funny variable names?) counts which column we are in. The Y-register holds the row within the column, and the X-register holds the LINARR index. The loop searches each column looking for the first populated square. When it finds one, the row index of the square is stored in the LV arrray. If it finds no populated square in the column, it assigns a value of 5 to the corresponding LV element. The sequence of CPX, BNE, LDX instructions in lines 4060-4220 translate the current row count in X into an index for LINARR and resume the loop. This is the clumsiest kind of code. It is special purpose code, code that is executed only once per condition. During program execution, much of the code is effectively useless, testing for conditions that do not exist. A more elegant solution is called for here. I was too lazy to be elegant; I just slopped the code together.

EVALUATING THE STRENGTH OF THE LINE (LPTS)

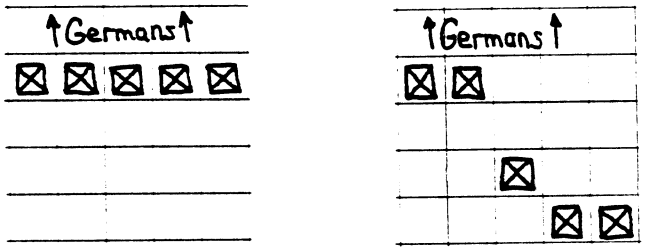
Now that the analytical tools we need are in place, we are ready to begin analysis of the position. We shall analyze the strength of a given line configuration by assigning points to it. We will assign various points for the various features we look for in a good line. These points will be stored in a variable called LPTS. Initially, we shall set this variable to zero and during the course of the evaluation we shall add to it or subtract from it.

The calculation begins on line 4240. We first evaluate the configuration for its completeness. Is there a unit in every single column in the array? For each populated column, we add $28 to LPTS (now in the accumulator). This is done in lines 4240-4320.

We then test if the contemplated presence of our army would fill an otherwise empty column. The test for this is simple and inelegant (lines 4360-4460). An easier way to have done this would have been LDA LV+2/CMP #$02/BNE Y95. It seems so simple and obvious now. In any event, if the condition is satisfied, we add $30 to LPTS.

We don't want to create a traffic jam, so we must evaluate the degree of blocking in this array. This is done by testing the frontmost unit in each column and looking behind it; if somebody is in that square the retreat route of the front unit and the attack route of the rear unit are both blocked. This is undesirable. Subtract $20 points for each such case (lines 4500-4730).

Our next concern is with penetrations. We do not want to create a line configuration which is easily flanked. A picture illustrates the problem.



GOOD BAD

The right arrangement is bad because it allows the enemy to easily penetrate to the rear of the most forward units before engaging the line. This places these forward units in jeopardy. We want a tight, parallel line as in the example on the left. It took me a lot of thinking to translate this concept into terms that the machine could execute. The final result was surprisingly easy to program. It is not so easy to explain. We have five columns in our square. We are going to take each column in turn, calling it OCOLUM, and compare its LV value with each of the other columns. While we are doing this test, we refer to the other column as COLUM. I know, the labelling seems backwards. Forgive me, I was feverish with effort. The comparison is made by subtracting the LV value of the one column from that of the other. If they are equal, there is no problem and we proceed to the next other column. If the latter column is more forward than the first, then we move to the next column; the discrepancy will be handled when the other column is directly tested. If the latter column is more rearward than the primary column, then a penalty must be extracted. The penalty I use is a power of two, one power for each row of discrepancy. The evaluation is done in lines 4880-4990.

We have now calculated the strength of the line and stored it in LPTS. However, the importance of this strength depends on the amount of danger coming from the direction in question. A line which is strong facing north will probably be weak to an attack from the west. We must therefore evaluate the strength of the line in light of the danger vector on the army. I do this by multiplying LPTS by the IFR value for the direction for which the line was evaluated. This multiplication is done in lines 5100-5370. The first 14 lines select the IFR to be used by some more inelegant code. The preparation for the multiplication is done in lines 5240-5280; the multiplication itself is done in lines 5290-5370. As with the long division,

this routine is a triumph of pedestrian programming. To multiply A by B, I add A to itself B times. It is a two-byte add, and only the upper byte (ACCHI) is important to me. I throw away the lower byte in the accumulator.

NEXT SECONDARY DIRECTION

I have now calculated the line configuration value of the square from one direction. I must now perform the same evaluation for each of the other directions. First I increment the secondary direction counter (SECDIR). Then I rotate the array. It is easier to rotate the array in place and evaluate it than to write code that can look from any direction. My code is customized to look at the 25-square array from the north. To look at it from other directions, I simply rotate it to those directions. This is done with an elegant piece of code (at last!) in lines 5480-5580. First I store the array LINARR into a temporary buffer array (BAKARR). Then I rotate it by a pointer array called ROTARR. This array holds numbers that tell where each array element goes when the array is rotated 90 degrees to the right. Thus, the zeroth element of ROTARR is a 4; that means that the zeroth element of LINARR should now be the fourth element. With the rotation done, the program flow loops back up to the beginning of this huge loop.

In developing this code I made heavy use of flowcharts. When I was satisfied with these I then wrote a small BASIC program that performed most of the manipulations in this chunk of code. It took only a few hours to write and test the BASIC code and verify that the fundamental algorithms would work as I had intended. Only then did I proceed to write the assembly code. This shows the value of BASIC: it is an excellent language for tossing ideas together and checking their function. I firmly believe that almost any assembly language project on a personal computer should have several BASIC tools developed just for supporting the effort. I wrote four different BASIC programs as part of the EASTERN FRONT development cycle. They are no longer useful, so I have discarded them.

EVALUATING IMMEDIATE COMBAT FACTORS

It is not good enough to analyze the danger in a square in terms of some obscure danger vector. It is also necessary to ask the simple question, how close is the nearest German unit? The proximity of a German unit will be of great significance to a Russian unit, although the precise significance will depend on whether the Russian is pursuing an offensive or a defensive strategy. In considering the direct combat significance of a square, we must also consider the defensive bonus provided by the terrain in the square.

These factors are considered in lines 5620-6310. After storing the modified line points value into SQVAL (square value), we determine the range to the nearest German unit. This is done with a straightforward loop that subtracts the coordinates of each German unit from the target square's coordinates, takes the absolute value, and adds the two results together. If the resulting range is less than the best previous value, it becomes the new best value (NBVAL).

This range to the closest German unit, when multiplied by the IFR, will give us the specific danger associated with the square. However, IFR is not a signed value; it is always positive. If the Russians are doing well, then IFR will be small but still positive. In such a case the value of IFR*NBVAL would be a measure of the opportunity presented to the Russian, not a measure of danger. Thus, small values of IFR demand that IFR*NBVAL be interpreted differently. The logic to do this is managed in lines 5930-6050. The IFR is subtracted from $F; if the result is greater than zero it is doubled and stored into TEMPR to act as a fake IFR; NBVAL is replaced by 9-NBVAL. The effect of these strange manipulations is to invert the meaning of the code about to be executed. This succeeding code was intended to determine the importance of running from a square. With the inversion, it will also determine the importance of attacking the same square.

The fooled code (lines 6090-6250) begins by checking the square to see if it is occupied by a German. If so, it immediately removes the square from consideration; we don't go around picking fights with Germans when we are the underdogs. Note that this will never happen when the Russians are using offensive strategy. If the square is unoccupied, we add the terrain bonus to NBVAL; this is a crude way of including terrain into the computation. I now think that this was not the correct way to handle terrain.

In lines 6200-6250 I execute one of my disgustingly familiar Neanderthal multiplications. I then add this value to SQVAL (lines 6270-6310).

TRAFFIC AND DISTANCE PENALTIES

The final tasks are to include penalties for traffic jams and long-distance marching. The former is necessary to make sure that Russians don't waste time crowding into the same square. The latter reflects the brutal reality that things sometimes do not go as expected, and so plans that call for armies to march long distances in the face of the enemy are seldom prudent.

The code for making these tests is simple (lines 6350-6870). The first test (lines 6350-6540) is a loop that tests all the other Russians, looking for one that has already chosen this square as an objective. If so, a penalty is extracted from SQVAL. The second test (lines 6580-6870) calculates the range from the army's current position to the target square. If it is greater than 6, the target is unreachable and the square is ruled out; SQVAL is set to zero. If not, 2 raised to the power of the range is subtracted from the SQVAL. With this work done, we have completed our calculation of the value of this square.

FINAL SQUARE EVALUATION

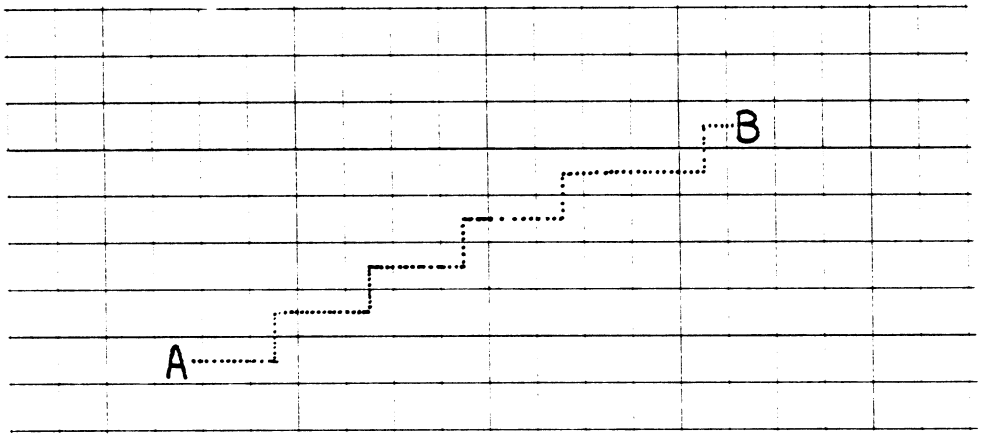
We now compare the value of this square with the best value we have obtained so far (lines 6910-6970). If our new value is better, it becomes the best. If not, we forget it. In either event, we go to the next square

and loop back to the far beginning (lines 6980-7020).

Upon completion of this gigantic process we have obtained a best square. In lines 7040-7150 we make this square our newest objective for this army. We then look at the START key to see if the human player has finished his move. If not, we continue our analysis, evaluating more and more squares without end. If so, we jump to a completely different section.

TRANSLATING TARGETS INTO ORDERS

If the human has pressed the START key, we must convert the targets figured by the previous routines into orders for execution in the mainline routine. This task in done in the remaining section of the module. The fundamental problem solved in the module is a very standard problem in computer graphics. It is depicted in the following diagram:



Starting at square A, what is the straightest path to square B? Specifically, what sequence of single steps will take you from A to B in the straightest possible line? For reasons of computational efficiency, we desire to find the answers without resorting to multiplications or divisions. The problem has been solved in its most general case, and the solution is so powerful that it is easily adapted to circles, ellipses, parabolas, and other curves. Unfortunately, I was unaware of this solution when I wrote these routines, so I had to make up my own, and thereby hangs a tale.

The obvious solution is to compute the slope of the line joining A and B, and then walk from A towards B, measuring the slope generated by each proposed step and comparing this resultant step with the desired slope; if the slope resulting from a proposed step is the closest that can be obtained, then that step is the best. Unfortunately, calculating a slope requires dividing a delta-y by a delta-x, and division is not allowed.

I found my solution in the calendraic system of the Mayan Indians. They never developed the concept of the fraction, and so they had a terrible time expressing the length of the year. Do you know how hard it is to measure the length of the year when you have no number for one-quarter day? They developed a novel solution: instead of declaring that one year is 365 and 1/4

days long, they declared that 4 years are 1461 days long. They refined the method to state that 25 years are 9131 days long. This procedure can be extended to arbitrary precision. Indeed, the Mayans did just that; their measurement of the length of the year was more accurate than the contemporary European value.

The basic idea of the technique is simple: your quantity is divided into a whole part and a fractional part. You can't get your hands on the fractional part, so you keep a running sum, adding both whole and fractional parts until the accumulated fractional parts add up to one; then the whole part will tick over an extra time. That's the event to watch for; it tells you what the fractional part is.

The first step in implementing this algorithm is to calculate some intermediate values. These are HDIR and HRNGE, the horizontal direction from A to B, and the horizontal range (delta-x). VDIR and VRNGE are the corresponding values for the vertical separation. These four values are calculated in lines 7370-7540.

Next we calculate the larger range LRNGE and the smaller range SRNGE, as well as the corresponding directions LDIR and SDIR (lines 7550-7690). Then we prepare some counting variables by setting them equal to zero: RCNT, the number of steps taken, and RORD1 and RORD2, the actual orders assigned. RANGE is the total distance from A to B in non-Pythagorean measure. CHRIS (I was getting desperate for variable names) is the rollover counter. I initialized myself to half of LRNGE.

We now begin the walk from A to B. On each step we shall assume that we should take a step in the larger direction (LDIR). In so doing we add SRNGE to CHRIS; if CHRIS exceeds RANGE then we must instead take a small step in direction SDIR. The figuring for this is done in lines 7830-7940. The code runs fast. The orders that result from this are folded into the main orders in lines 8110-8140, another case of that weird code that first popped up in the interrupt module. If you didn't figure it out then, you might as well figure it out now.

A few more manipulations loop back to finish the walk to point B; then the army's orders are stored and the next army is given its orders until all armies have been taken care of. With that, the routine is complete and it returns to the mainline routine in line 8340. That was simple enough, wasn't it?

NARRATIVE HISTORY

A common misconception among non-programmers is that a program is a static product, something that springs complete from the hand of the programmer. What they do not realize is that a truly original program like EASTERN FRONT 1941 does not leap out of the programmer's mind and into the computer. It starts with an inspiration, a vision that sketches the outlines broad and clear but leaves the individual brushstrokes undefined. The programmer labors long and hard to translate this vision into a cybernetic reality. But the process of converting the pastels and soft shades of the vision into the hard and sharp lines of machine code inevitably produces contradictions between the fine details. As many small ideas crystallize into a single whole, mismatches and discord are frequent. The programmer flits over the design, rearranging ideas, polishing rough edges, and reconciling differences. In this process many of the original ideas are warped and twisted until they no longer resemble their original forms. It is very easy, on examining a program closely, to unearth many of these convoluted elements and conclude that the programmer lacks common sense. In truth, the only way to understand a program is to follow its evolution from start to finish. I have tried to explain some of the odder aspects of this program in terms of historical happenstance. In this essay I will narrate the history of the entire project. I hope that this will make the final product more understandable.

ORIGINS

EASTERN FRONT (1941) began as OURRAH POBIEDA in June of 1979. The original name is Russian for "Hooray for the Motherland!" and was the Russian war cry. It was retained until the last minute; I was finally convinced that the simpler name would sell better.

OURRAH POBIEDA was initially conceived as a division-level game of combat on the Eastern Front. The emphasis of the design was on the operational aspects of combat in that campaign. I wanted to demonstrate the problems of handling division-sized units. The design placed heavy emphasis on mechanical aspects of warfare. Thus, it had strong logistics and movement features. It also had a major subsystem dealing with operational modes. The player could place each unit into one of many different modes such as movement, assault, reconnaissance in force, probing assault, and so on. Each mode had different combinations of movement capabilities, attack strength, and defense strength. There was also a provision for the facing of a unit that allowed flanking attacks.

I wrote the program in BASIC on a PET computer in May and June of 1979. When I got the program up and running on the machine, I quickly realized that I had a dog on my hands. The game had many, many flaws. There were good ideas in it---the logistics routines, the combat system, and the movement system were all very good. But the game as a whole did not work. It was dull, confusing, and slow. I wisely consigned all of my work into a file folder and started on a new design. Someday, when I had shaken off whatever preconceptions were contaminating my mind, I would come back to the game and start over with a fresh outlook.

REBIRTH

Fifteen months passed. I went to work for Atari, programming first on the Video Computer System and then on the Home Computer. In September of 1980 I saw a program written by Ed Rothberg of Atari that finely scrolled the text window. It was a short demo that did nothing other than move the characters around, but it shouted out its potential. I showed it to several other wargame designers and pointed out its implications for our craft. They listened politely but did not act on my suggestion that they use the capability.

Several weeks later I began exploring the fine scrolling capabilities of the machine myself. I took apart Ed's code and wrote a new routine that was more useful for me. I then generalized this routine to produce SCRL19.ASM, a demonstration scrolling module. This module has been spread around in an effort to encourage programmers to use the scrolling. By mid-November I had completed SCRL19.ASM and was finishing up another wargame project. I was beginning to think about my next project. I decided it was time to pull out all the stops and do a monster game, a game with everything. It would be a 48K disk-based game with fabulous graphics. It seemed obvious that the Eastern Front was the ideal stage for such a game. I therefore began planning this new game. In the meantime, I began converting SCRL19.ASM to produce a map of Russia. This map was completed on December 10. It impressed many people, but it was only a map; it didn't do anything other than scroll.

DESIGNING A NEW GAME

Game design is art, not engineering. During December I took many long walks alone at night, sorting through my thoughts and trying to formulate my vision of the game clearly. I sifted through all of my old documents on the PET version of OURRAH POBIEDA, trying to glean from that game the essence of all that was good and all that was bad. Mostly, I thought about what it would be like to play the game. What will go through the head of a person playing my game? What will that person experience? What will he think and feel?

During all this time I never once put pencil to paper to record my thoughts. I deliberately avoided anything that would constrain creative flights of fancy. I also fought off the urge to rush the job. I spent four weeks just thinking. I didn't want to start designing a game that wasn't fully conceived yet.

Then, in January, the vision was clear. I knew (or thought I knew) exactly what the game would be like. I wrote a one-page description of the game. The original document is reproduced at the end of this essay. You will note that it is a surpisingly accurate description of the final product. Also note what is specified. The information given represents my judgment of the critical design and technical factors that would dominate the development

of the game. Note especially the importance I attached to human interface and graphics. This reflects my belief that computation is never a serious problem, but interface is always the primary problem.

PLUNGING INTO THE MORASS

I now began the serious task of implementing the design. At first I proceeded slowly, cautiously. I documented all steps carefully and wrote code conservatively. I didn't want to trap myself with inflexible code at an early stage of the game. First I rewrote the map routine, which involved the data module and the interrupt module. (I decided at the outset that I would need separate modules, as I fully expected the entire program to be too big to fit in one module.) As part of this effort I redesigned the display list and display list interrupt structure. This gave me a much better display. By this time, early February, I was in full gear and was working nights and weekends, perhaps 20 hours per week. I made last changes in the character sets and nailed down the map contents. Next came the unit displays. I wrote the swapping routine and began putting units on the map. They couldn't move or do anything, but they sure looked pretty.

In late February I began work on the input routines. So far everything had gone in smoothly. There had been a lot of work, but most routines had worked properly on the first or second try. My first real headache came when I tried to design the input routines. I had decided that most of the game would be playable with only the joystick. The player would use the START key to begin a move, but otherwise the keyboard was to be avoided. I hung up on the problem of cancelling orders. There seemed to be no way to do it with the joystick. This caused me great consternation. I finally gave in and used the SELECT key for cancelling orders. This may surprise you, for the final product uses the space bar and the initial spec clearly states that space bar would be used. I didn't want to use the keyboard, so I insisted on using the yellow buttons. My playtesters (most notably Rob Zdybel) convinced me to go back to the space bar.

My next problem with the input routines arose when I tried to display a unit's existing orders. I had no end of problems here. My original idea had been to use player-missile graphics to draw some kind of dotted path that would show a unit's planned route instantly. Unfortunately, there weren't enough players and missiles to do the job properly. It could only be done if I used single dots for each square entered. I put the display up on the screen and decided that it did not look good enough. So it was back to the drawing board. The solution I eventually came up with (after considerable creative agony) is the system now used---the moving arrow that shows a unit's path. This takes a little longer but the animation effect is nice.

THE LIGHT AT THE END OF THE TUNNEL

By now it was early March and I paused to consider the pace of the effort. I could see how much effort would be needed to complete the task. I listed each of the remaining tasks and estimated the amount of time necessary

for each. I then realized that the program would not be finished until late June. This was an unpleasant surprise, for I had been planning all along to unveil the game at the ORIGINS wargaming convention over the 4th of July weekend. The schedule appeared to give me very little extra time in the event of problems. I did not like the looks of it. I resolved to redouble my efforts and try to get ahead of the schedule.

MAINLINE MODULE

With the input routines done it was time to work on the mainline module. The very first task was to take care of calendraic functions. I wrote the routines to calculate the days and months; this was easy. Next came the tree color changes with seasons; this was also easy. The first problem developed with the freezing of rivers and swamps during the winter. I was unable to devise a simple way of doing this. I plunged into the problem with indecent haste and threw together a solution by force of effort. The result was impressive, but I'm not sure I did the right thing. It cost me a week of effort, no great loss, and a lot of RAM, which at the time seemed inconsequential because I was still planning on the game taking 48K of memory. Later, when I chose to drop down to 16K, I found myself cramped for RAM, and the expenditure of 120 bytes began to look wasteful.

Fortunately, I emerged from these problems unscathed. I was not tired yet, the project seemed on track and my morale was still high. Morale is important---you can't do great work unless you are up for it.

The next task was movement execution. This went extremely well. I had planned on taking two weeks to get units moving properly; as luck would have it, the routines were working fine after only one week. I was hot!

COMBAT ROUTINES

As March ended, I was beginning work on the combat resolution routines. I had some severe problems here. My routines were based closely on the systems used for the original OURRAH POBEIDA. After some thought, I began to uncover serious conceptual problems with this system. A combat system should accomplish several things. It should provide for attrition due to the intensity of combat. It should also provide for the collapse of a unit's coherence and its subsequent retreat. The routines I had were too bloody. They killed many troops by attrition but did not retreat units readily. I analyzed them closely and concluded that the heart of the problem lay in the fact that combat was completely resolved in a single battle. From this I came up with the idea of the extended battle covering many movement subturns during the week. By stretching out the battle in this way I was able to solve the problem and achieve a much better combat system. I still retained the central idea of the earlier system, which broke a unit's strength up into muster strength and combat strength.

ARTIFICIAL INTELLIGENCE

In early April I turned to the last major module of the project: the artificial intelligence routines. This module frightened me, for I was unsure how to handle it. Looking back, I cannot believe that I invested so much time in this project in the blithe expectation that the artificial intelligence routines would work out properly. I threw myself into them with naive confidence. I carefully listed all of the factors that I wanted the Russian player to consider. Then I prepared a flowchart for the sequence of computations. This flowchart was subsequently rewritten many times as I changed the design.

My biggest problems came with the method of analyzing the robustness of the Russian line. My first approach was based on the original OURRAH POBEIDA method. I started at one end of the line and swept down the line looking for holes. When a hole was found I marked it and jumped onward to the other side of the hole. When the line was fully traced I sent reinforcements to the holes and weak spots in the line. This worked in OURRAH POBEIDA but would not work in the new program. The Russian line in the new program would be far more ragged than in the original game. In some places, the holes would be bigger than the line. In such cases, the algorithm would almost certainly break down.

A new algorithm was required. After many false starts, I came up with the current scheme, which broke the line up into small segments 5 squares wide. This 5-square chunk is then applied to each unit in the Russian army, providing a kind of moving average to smooth the line and bind together the different units in the line. I am very proud of this design, for it is quite flexible and powerful in its ability to analyze a line structure. An interesting aspect of this design is that I originally designed it to handle a smaller segment only three squares wide. After the code had been written, entered, and partially debugged I decided that it would work better with a 5-square width. I modified the code to handle the new width in a few days. The transition was really quite clean. This indicates that I wrote the original code very well, for the ease with which code can be rewritten is a good measure of its quality.

FIRST STARTUP

It was now mid-May. Six months had passed since I had begun the first efforts on the game. One evening, rather late, I finished work on the artificial intelligence routine and prepared to actually play the game for the first time. Many, many times I had put the game up to test the performance of the code, but this was the first time I was bringing the game up solely to assess the overall design. Within ten minutes I knew I had a turkey on my hands. The game was dull and boring, it took too much time to play, it didn't seem to hang together conceptually, and the Russians played a very stupid game.

THE CRISIS

I remember that night very well. I shut off the machine and went for a long walk. It was time to do some hard thinking. The first question was, can the game be salvaged? Are the problems with this game superficial or fundamental to the design? I decided that the game suffered from four problems: There were too many units for the human to control. The game would require far too long to play. The game was a simple slugfest with little opportunity for interesting ploys for the German. The Russians were too stupid. The second question I had to answer was, should I try to maintain my schedule, or should I postpone the game and redesign it?

That was a long night. One thing kept my faith: my egotism. Most good programmers are egomaniacs, and I am no exception. When the program looked hopeless, and the problems seemed insurmountable, one thing kept me going---the absolute certainty that I was so brilliant that I could think up a solution to any problem. Deep down inside, every good programmer knows that the computer will do almost anything if only it is programmed properly. The battle is not between the programmer and the recalcitrant computer; it is between the programmer's will and his own stupidity. Only the most egotistical of programmers refuses to listen to the "I can't do it" and presses on to do the things which neither he nor anybody else thought possible. But in so doing, he faces many lonely nights staring at the ceiling, wondering if maybe this time he has finally bitten off more than he can chew.

I threw myself at the task of redesigning the program. First, I greatly reduced the scale of the program. I had intended the game to cover the entire campaign in the east from 1941 to 1945. I slashed it down to only the first year. That suddenly reduced the projected playing time from a ridiculous 12 hours to a more reasonable 3 hours. I then drastically transformed the entire game by introducing zones of control. Before then units were free to move through gaps in the line at full speed. This single change solved a multitude of problems. First, it allowed me to greatly reduce the unit count on both sides. One unit could control far more territory now, so fewer units were necessary. With fewer units, both players could plan their moves more quickly. Second, Russian stupidity was suddenly less important. If the Russians left small holes in the line, they would be covered by zones of control. Third, it made encirclements much easier to execute, for large Russian forces could be trapped with relatively few German armored units.

My third major change to the game design was the inclusion of logistics. I had meant to have supply considerations all along, but I had not gotten around to it until this time. Now I put it in. This alone made a big change in the game, for it permitted the German to cripple Russian units with movement instead of combat. Indeed, the encirclement to cut off supplies is the central German maneuver of the entire game.

It was about this time that I also committed to producing a game that

would run on a 16K system. I had suspected since April that the entire program would indeed fit into 16K but I did not want to constrain myself, so I continued developing code with little thought to its size. Yet it is hard to deny one's upbringing. I had learned micros on a KIM with only 1K of RAM, later expanded to 5K. I had written many of my early programs on a PET with 8K of RAM, later 16K. I had written programs at Atari to run in 16K. My thoughts were structured around a 16K regime. When the first version of the program ran in May, it fit in almost exactly 16K. I never took anything out to meet the 16K requirement; I simply committed to maintaining the current size.

FRANTIC JUNE

During the first two weeks of June I worked like a madman to implement all of these ideas. The program's structure went to hell during this time. I was confident of what I was doing, and was willing to trade structure for time. I had all the changes up and running by mid-June. It was then that I released the first test version to my playtesters. I also began the huge task of polishing the game, cleaning out the quirks and oddities. This consumed my time right up to the ORIGINS convention on July 3-5. We showed the game to the world then, and it made a favorable impression on those who saw it. The version shown there was version 272. It was a complete game, and a playable game, and even an enjoyable game. It was not yet ready for release.

THE POLISHING STAGES

Two of the most critical stages in the development of a program are the design stage and the polishing stage. In the former, the programmer is tempted to plunge ahead without properly thinking through what he wants to achieve. In the latter, the programmer is exhausted after a major effort to complete the program. The program is now operational and could be released; indeed, people are probably begging for it immediately. The temptation to release it is very strong. The good programmer will spurn the temptation and continue polishing his creation until he knows that it is ready to be released.

Polishing occupied my attentions for six weeks. I playtested the game countless times, recording events that I didn't like, testing the flow of the game, and above all looking for bugs. I found bugs, too. One by one, I expurgated them. I rewrote the zone of control routine to speed it up and take less memory. I made numerous adjustments in the artificial intelligence routines to make the Russians play better. Most of my efforts were directed to the timing and placement of reinforcements. I found that the game was balanced on a razor-edge. A good player would have victory within his reach right up through December, but then the arrival of a large block of Russian reinforcements would dash his chances. I spent a great deal of time juggling reinforcements to get the game tightly balanced.

During this time playtesters were making their own suggestions for the

game. Playtesters are difficult to use properly. At least 90 percent of the suggestions they make are useless or stupid. This is because they do not share the vision that the designer has, so they try to take the game in very different directions. The tremendous value of playtesters lies in that small 10 percent that represents valuable ideas. No designer can think of everything. Every designer will build personal quirks into a game that can only hurt the design. The playtesters will catch these. The good designer must have the courage to reject the bad 90 percent, and the wisdom to accept the good 10 percent. It's a tough business.

DELIVERY AND AFTERMATH

I delivered the final product to Dale Yocum at the Atari Program Exchange around the 20th of August. It was the 317th version of the program. The program went on sale 10 days later. It has generated favorable responses. I was not able to embark on a new project for ten weeks; I was completely burned out. I do not regret burning myself out in this way; anything less would not have been worth the effort.

Eastfront Game Preliminary Description.

Map: 64x64 squares

Unit count: 32 German corps
up to 64 Russian armies

Time scale: "Semi-time" of one week/turn. German enters moves for the next week (meanwhile, computer figures Russian move). When ready, move proceeds in real time.

Human interface: Map window on screen. ~~[illegible]~~ Joystick scrolls map + players. Putting unit under crosshairs, activates it, arrows show Then holding down button while twiddling joystick enters next order. arrows (players) pop onto screen showing orders. Space bar clears orders. Releasing button resumes scrolling.

START button starts ~~[illegible]~~ turn.

Colors:

~~Brown~~ background		Brown	
	PF0 1	Green	(forests, ~~swamps~~) DLI to ~~[illegible]~~ orange mountains
	PF1 ~~[illegible]~~ 3	Blue	(rivers, lakes, seas)
	PF2 0	Grey	(Germans, cities)
	PF3 ~~[illegible]~~ 2	Red	(Russians)
	PØ-P3	Pink	(arrows)
	MØ-M3	"	
DLIs	borders		
	red-orange text window		

[not enough color! Use dli's or time-multiplexed color.

CHARACTER SET DESCRIPTIONS

There are three character sets used in EASTERN FRONT 1941. The first is the standard text character set. The other two are graphics character sets used for the display of the map. These character sets allow 64 distinct characters in each set; each character can be presented in one of four colors. The two charts that follow give the critical assignments of characters in the character sets. I do not include the actual bit assignments for each character, as this information is not of primary interest to a designer.

Each chart gives the 6-bit number, which is the number that specifies the shape of the character, and the 8-bit number, which specifies the combination of color and shape that is used in the program. There are a few exceptions. For example, the river characters are normally presented in blue, but during winter they are presented in white to indicate that they have frozen. The character value must be changed to accomplish this. Another case is the solid character, which is normally white for the boundaries. It is also used in its red incarnation to show that a Russian unit has been attacked.

The character descriptions are also cryptic. The river characters are described in terms of the sides of the squares through which the river passes. For this usage, 1 means north, 2 east, 3 south, and 4 west. Thus, a 13 river goes from the northern edge of the square to the southern edge, while a 23 river goes from the eastern edge to the southern edge. River junctions are specified by the three edges that contain rivers.

Coastlines are specified in a similar fashion, with an additional convention. Coastlines are specified directionally, with the land on the right side of the path drawn. For example, a 24 coastline runs from the east side of the square to the west side, with the land on the north and the sea on the south. A 42 coastline would be similar with the land and sea on opposite sides.

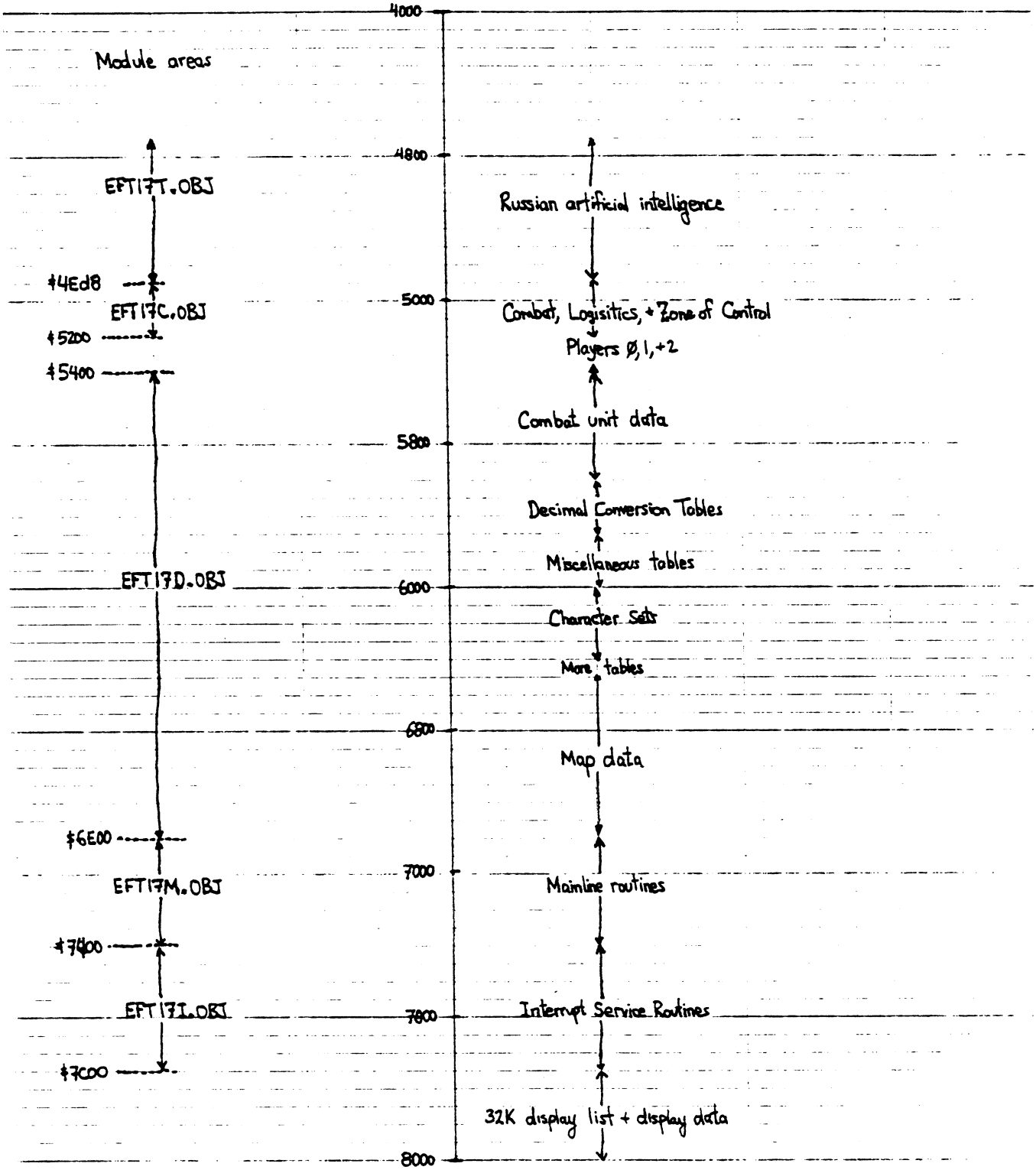
NORTHERN CHARACTER SET SUMMARY

6-BIT #	DESCRIPTION	8-BIT #	6-BIT #	DESCRIPTION	8-BIT #
0	clear	0	32	river 24	160
1	forest	1	33	river 24	161
2	forest	2	34	river 24	162
3	forest	3	35	river 24	163
4	forest	4	36	river 34	164
5	forest	5	37	river 34	165
6	forest	6	38	river 34	166
7	city	71	39	river 34	167
8	city	72	40	river 134	168
9	city	73	41	coastline 31	169
10	city	74	42	coastline 31	170
11	swamp	139	43	coastline 31	171
12	swamp	140	44	coastline 42	172
13	swamp	141	45	coastline 42	173
14	swamp	142	46	coastline 42	174
15	river 12	143	47	coastline 21	175
16	river 12	144	48	coastline 41	176
17	river 12	145	49	coastline 32	177
18	river 12	146	50	coastline 34	178
19	river 13	147	51	coastline 12	179
20	river 13	148	52	Finnish coastline	180
21	river 13	149	53	Finnish coastline	181
22	river 13	150	54	Finnish coastline	182
23	river 13	151	55	Finnish coastline	183
24	river 13	152	56	Finnish coastline	184
25	river 14	153	57	Finnish coastline	185
26	river 14	154	58	Lake Peipus	186
27	river 14	155	59	estuary 1	187
28	river 23	156	60	estuary 2	188
29	river 23	157	61	infantry	125 or 253
30	river 23	158	62	armor	126 or 254
31	river 24	159	63	solid	191

SOUTHERN CHARACTER SET SUMMARY

6-BIT #	DESCRIPTION	8-BIT #	6-BIT #	DESCRIPTION	8-BIT #
0	clear	0	32	river 34	160
1	mountain	1	33	river 34	161
2	mountain	2	34	river 124	162
3	mountain	3	35	Kerch straits	163
4	mountain	4	36	coastline 13	164
5	mountain	5	37	coastline 24	165
6	mountain	6	38	coastline 24	166
7	city	71	39	coastline 24	167
8	city	72	40	coastline 21	168
9	city	73	41	coastline 21	169
10	city	74	42	coastline 14	170
11	swamp	139	43	coastline 14	171
12	swamp	140	44	coastline 14	172
13	swamp	141	45	coastline 41	173
14	swamp	142	46	coastline 41	174
15	river 12	143	47	coastline 23	175
16	river 12	144	48	coastline 23	176
17	river 12	145	49	coastline 23	177
18	river 12	146	50	coastline 32	178
19	river 13	147	51	coastline 32	179
20	river 13	148	52	coastline 34	180
21	river 13	149	53	coastline 34	181
22	river 14	150	54	Crimea	182
23	river 14	151	55	Crimea	183
24	river 23	152	56	Crimea	184
25	river 23	153	57	Crimea	185
26	river 24	154	58	estuary 1	186
27	river 24	155	59	estuary 2	187
28	river 24	156	60	estuary 3	188
29	river 24	157	61	infantry	125 or 253
30	river 34	158	62	armor	126 or 254
31	river 34	159	63	solid	191

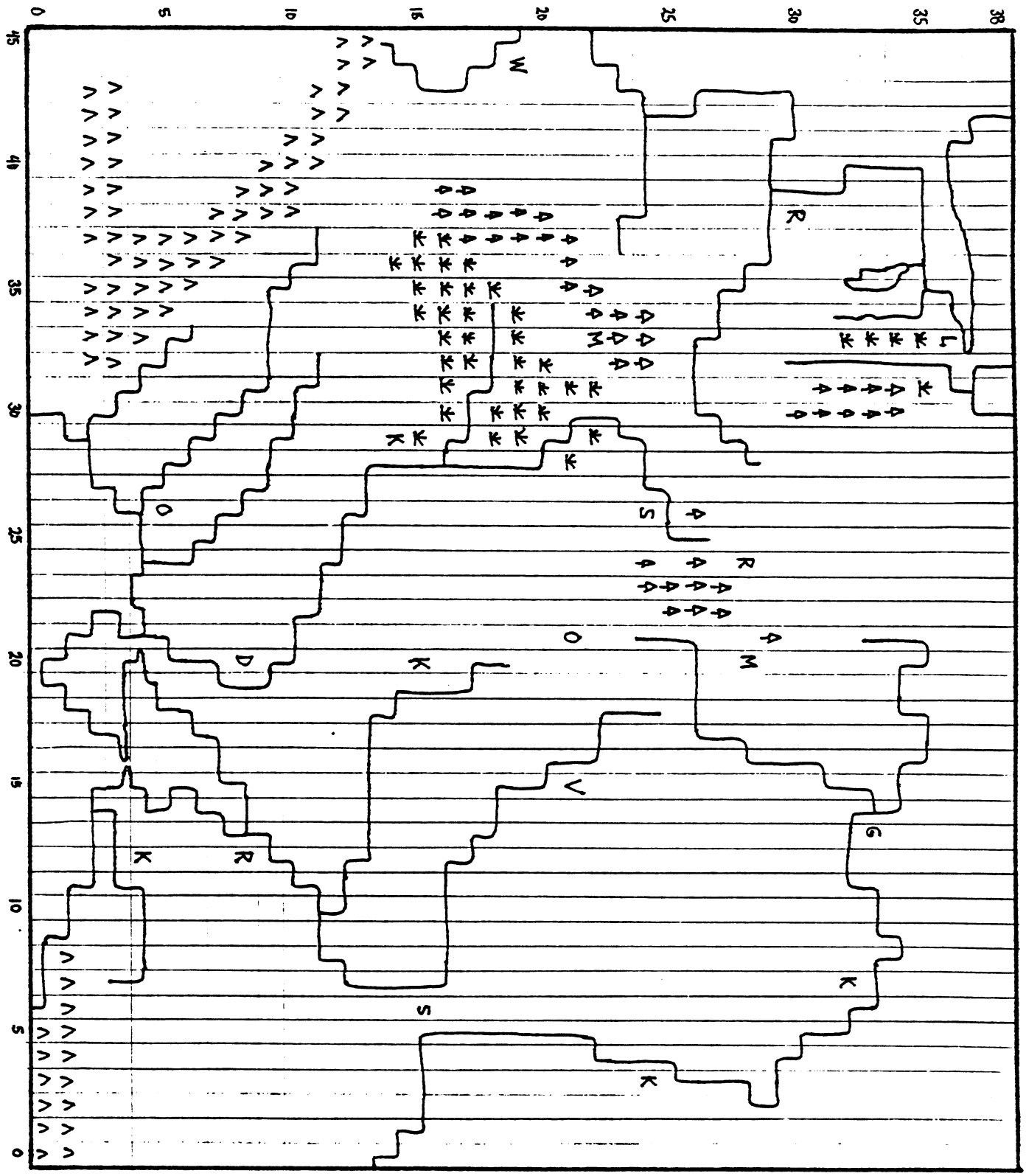
Final Memory Map, Eastern Front Scale is 1 division = 256 bytes Diskette version (Subtract $4000 for cassette)



Other Areas:

Page Zero: $BØ → $CE

Page Six: $600 → $[redacted]6FF


0
5
10
15
20
25
30
35
38
45
40
35
30
25
20
15
10
5
0
W
R
L
M
S
O
D
K
V
G

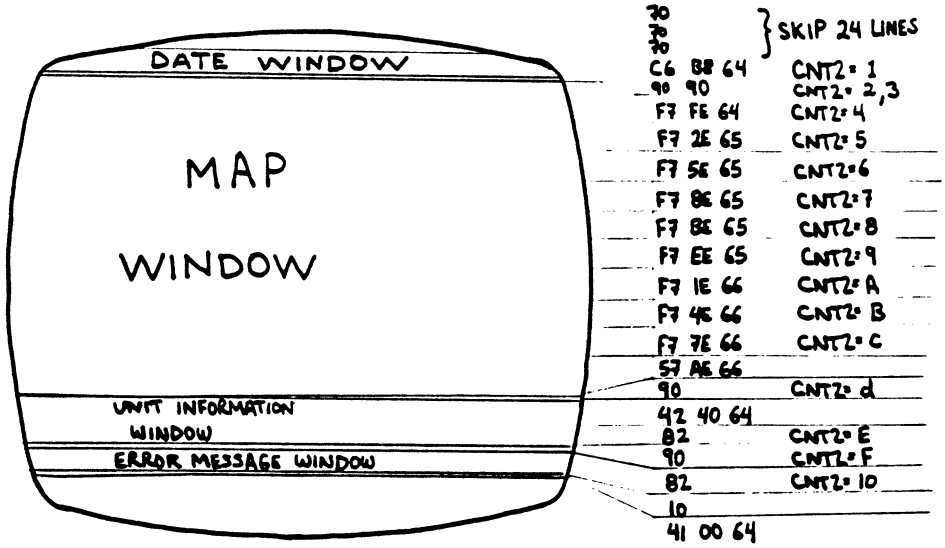
UNIT CHARACTERISTICS

SEQUENCE #		NAME	CORPSX	CORPSY	MSTRNG	SWAP	ARRIVE	CORPT
0	0	INFANTRY CORPS	0	0	0	0	255	0
1	24	PANZER CORPS	40	20	203	126	0	3
2	39	PANZER CORPS	40	19	205	126	255	3
3	46	PANZER CORPS	40	18	192	126	0	3
4	47	PANZER CORPS	40	17	199	126	0	3
5	57	PANZER CORPS	40	16	184	126	0	3
6	5	INFANTRY CORPS	41	20	136	125	0	0
7	6	INFANTRY CORPS	40	19	127	125	0	0
8	7	INFANTRY CORPS	41	18	150	125	0	0
9	8	INFANTRY CORPS	41	17	129	125	0	0
10	9	INFANTRY CORPS	41	16	136	125	0	0
11	12	INFANTRY CORPS	42	20	109	125	255	0
12	13	INFANTRY CORPS	42	19	72	125	255	0
13	20	INFANTRY CORPS	42	18	70	125	255	0
14	42	INFANTRY CORPS	42	17	81	125	255	0
15	43	INFANTRY CORPS	43	19	131	125	255	0
16	53	INFANTRY CORPS	43	18	102	125	255	0
17	3	ITAL INF CORPS	43	17	53	125	255	64
18	41	PANZER CORPS	41	23	198	126	0	3
19	56	PANZER CORPS	40	22	194	126	0	3
20	1	INFANTRY CORPS	40	21	129	125	0	0
21	2	INFANTRY CORPS	41	21	123	125	0	0
22	10	INFANTRY CORPS	41	22	101	125	0	0
23	26	INFANTRY CORPS	42	22	104	125	0	0
24	28	INFANTRY CORPS	42	23	112	125	0	0
25	38	INFANTRY CORPS	42	24	120	125	0	0
26	3	PANZER CORPS	40	15	202	126	0	3
27	14	PANZER CORPS	41	14	195	126	0	3
28	48	PANZER CORPS	42	13	191	126	0	3
29	52	PANZER CORPS	41	15	72	126	255	3
30	49	INFANTRY CORPS	42	14	140	125	0	0
31	4	INFANTRY CORPS	42	12	142	125	0	0
32	17	INFANTRY CORPS	43	13	119	125	0	0
33	29	INFANTRY CORPS	41	15	111	125	0	0
34	44	INFANTRY CORPS	42	16	122	125	255	0
35	55	INFANTRY CORPS	43	16	77	125	255	0
36	1	RUM INF CORPS	30	2	97	125	0	48
37	2	RUM INF CORPS	30	3	96	125	0	48
38	4	RUM INF CORPS	31	4	92	125	0	48
39	11	INFANTRY CORPS	33	6	125	125	0	0
40	30	INFANTRY CORPS	35	7	131	125	0	0
41	54	INFANTRY CORPS	37	8	106	125	0	0
42	2	FINN INF CORPS	35	38	112	125	0	32
43	4	FINN INF CORPS	36	37	104	125	0	32
44	6	FINN INF CORPS	36	38	101	125	255	32
45	40	PANZER CORPS	45	20	210	126	2	3
46	27	INFANTRY CORPS	45	15	97	125	255	0
47	1	HUN PZR CORPS	38	8	98	126	2	83
48	23	INFANTRY CORPS	45	16	95	125	5	0
49	5	RUM INF CORPS	31	1	52	125	6	48
50	34	INFANTRY CORPS	45	20	98	125	9	0
51	35	INFANTRY CORPS	45	19	96	125	10	0
52	4	ITAL INF CORPS	32	1	55	125	11	64
53	51	INFANTRY CORPS	45	17	104	125	20	0
54	50	PZR GRNDR CORPS	45	18	101	126	24	7

SEQUENCE #		NAME	CORPSX	CORPSY	MSTRNG	SWAP	ARRIVE	CORPT
55	7	MILITIA ARMY	29	32	100	253	4	4
56	11	MILITIA ARMY	27	31	103	253	5	4
57	41	INFANTRY ARMY	24	38	110	253	7	0
58	42	INFANTRY ARMY	23	38	101	253	9	0
59	43	INFANTRY ARMY	20	38	92	253	11	0
60	44	INFANTRY ARMY	15	38	103	253	13	0
61	45	INFANTRY ARMY	0	20	105	253	7	0
62	46	INFANTRY ARMY	0	8	107	253	12	0
63	47	INFANTRY ARMY	0	18	111	253	8	0
64	48	INFANTRY ARMY	0	10	88	253	10	0
65	9	TANK ARMY	0	14	117	254	10	1
66	13	TANK ARMY	0	33	84	254	14	1
67	14	TANK ARMY	0	11	109	254	15	1
68	15	TANK ARMY	0	15	89	254	16	1
69	16	TANK ARMY	0	20	105	254	18	1
70	7	CAV ARMY	0	10	93	254	7	2
71	2	TANK ARMY	21	28	62	254	0	1
72	19	INFANTRY ARMY	21	27	104	253	0	0
73	18	INFANTRY ARMY	30	14	101	253	0	0
74	1	CAV ARMY	30	13	67	254	0	2
75	27	INFANTRY ARMY	39	28	104	253	0	0
76	10	TANK ARMY	38	28	84	254	0	1
77	22	INFANTRY ARMY	23	31	127	253	0	0
78	21	INFANTRY ARMY	19	24	112	253	0	0
79	13	INFANTRY ARMY	34	22	111	253	0	0
80	6	TANK ARMY	34	21	91	254	0	1
81	9	MILITIA ARMY	31	34	79	253	0	4
82	2	INFANTRY ARMY	27	6	69	253	0	0
83	1	MILITIA ARMY	33	37	108	253	0	4
84	8	INFANTRY ARMY	41	24	118	253	0	0
85	11	INFANTRY ARMY	40	23	137	253	0	0
86	1	TANK ARMY	39	23	70	254	0	1
87	7	TANK ARMY	42	25	85	254	0	1
88	3	INFANTRY ARMY	39	20	130	253	0	0
89	4	INFANTRY ARMY	39	22	91	253	0	0
90	10	INFANTRY ARMY	39	18	131	253	0	0
91	5	TANK ARMY	39	17	71	254	0	1
92	8	TANK ARMY	39	21	86	254	0	1
93	3	CAV ARMY	37	20	75	254	0	2
94	6	CAV ARMY	39	19	90	254	0	2
95	5	INFANTRY ARMY	39	16	123	253	0	0
96	6	INFANTRY ARMY	39	15	124	253	0	0
97	12	INFANTRY ARMY	40	14	151	253	0	0
98	26	INFANTRY ARMY	41	13	128	253	0	0
99	3	TANK ARMY	41	12	88	254	0	1
100	4	TANK ARMY	39	11	77	254	0	1
101	11	TANK ARMY	36	9	79	254	0	1
102	5	CAV ARMY	34	8	80	254	0	2
103	9	INFANTRY ARMY	32	6	126	253	0	0
104	12	TANK ARMY	35	9	79	254	0	1
105	4	CAV ARMY	30	4	91	254	0	2
106	2	CAV ARMY	28	2	84	254	0	2
107	7	INFANTRY ARMY	25	6	72	253	1	0
108	2	MILITIA ARMY	29	14	86	253	1	4
109	14	INFANTRY ARMY	32	22	76	253	1	0

SEQUENCE #		NAME	CORPSX	CORPSY	MSTRNG	SWAP	ARRIVE	CORPT
110	4	MILITIA ARMY	33	36	99	253	1	4
111	15	INFANTRY ARMY	26	23	67	253	1	0
112	16	INFANTRY ARMY	21	8	78	253	2	0
113	20	INFANTRY ARMY	29	33	121	253	2	0
114	6	INFANTRY ARMY	0	28	114	253	2	0
115	24	INFANTRY ARMY	28	30	105	253	3	0
116	40	INFANTRY ARMY	21	20	122	253	3	0
117	29	INFANTRY ARMY	21	28	127	253	4	0
118	30	INFANTRY ARMY	21	33	129	253	4	0
119	31	INFANTRY ARMY	20	27	105	253	5	0
120	32	INFANTRY ARMY	20	30	111	253	5	0
121	33	INFANTRY ARMY	12	8	112	253	6	0
122	37	INFANTRY ARMY	0	10	127	253	6	0
123	43	INFANTRY ARMY	0	32	119	253	7	0
124	49	INFANTRY ARMY	0	11	89	253	8	0
125	50	INFANTRY ARMY	0	25	108	253	8	0
126	52	INFANTRY ARMY	0	12	113	253	8	0
127	54	INFANTRY ARMY	0	23	105	253	9	0
128	55	INFANTRY ARMY	0	13	94	253	9	0
129	1	GD CAV ARMY	21	29	103	254	5	114
130	34	INFANTRY ARMY	25	30	97	253	5	0
131	1	GD INF ARMY	0	31	108	253	2	112
132	2	GD INF ARMY	0	15	110	253	9	112
133	3	GD INF ARMY	0	27	111	253	10	112
134	4	GD INF ARMY	0	17	96	253	10	112
135	39	INFANTRY ARMY	0	25	109	253	6	0
136	59	INFANTRY ARMY	0	11	112	253	11	0
137	60	INFANTRY ARMY	0	23	95	253	5	0
138	61	INFANTRY ARMY	0	19	93	253	17	0
139	2	GD CAV ARMY	0	21	114	254	2	114
140	1	TANK ARMY	0	33	103	254	11	1
141	1	GD TANK ARMY	0	28	107	254	20	113
142	5	GD INF ARMY	0	13	105	253	21	112
143	2	TANK ARMY	0	26	92	254	22	1
144	6	GD INF ARMY	0	10	109	253	23	112
145	3	TANK ARMY	0	29	101	254	24	1
146	4	TANK ARMY	0	35	106	254	26	1
147	38	INFANTRY ARMY	0	27	95	253	28	0
148	36	INFANTRY ARMY	0	15	99	254	30	0
149	35	INFANTRY ARMY	38	30	101	253	2	0
150	28	INFANTRY ARMY	21	22	118	253	3	0
151	25	INFANTRY ARMY	12	8	106	253	3	0
152	23	INFANTRY ARMY	20	13	112	253	3	0
153	17	INFANTRY ARMY	21	14	104	253	3	0
154	8	MILITIA ARMY	20	28	185	253	6	4
155	10	MILITIA ARMY	15	3	108	253	6	4
156	3	MILITIA ARMY	21	3	94	253	4	4
157	5	MILITIA ARMY	20	3	102	253	4	4
158	6	MILITIA ARMY	19	2	98	253	4	4
159	0	INFANTRY ARMY	0	0	0	255	32	0

DISPLAY LIST INTERRUPT SEQUENCING

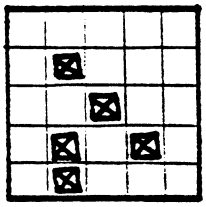


CNTR value	Register changed					
	CHBAS	COLBAK	COLPFØ	COLPF1	COLPF2	COLPF3
0 (vert. blank)	E0	[illegible] BO	[illegible] 6A	0C	94	46
1	60	1A	TRCOLR	–	–	–
3	–	EARTH	–	–	–	–
3≤CNT1≤d	62	–	28	–	–	–
d	E0	–	–	–	22	–
E	–	8A	–	–	–	–
F	–	–	–	00	3A	–
10	–	d4	–	–	–	–

Point System for Artificial Intelligence

A. Line Points - LPTS

(Values for this example)



LINARR = 0,0,0,0,0,0,M1,0,M2,M3
0,0,M4,0,0,0,0,0,M5,0
0,0,0,0,0

LV = 5,1,2,3,5

+ 40 points for each occupied column — LPTS = 120
+ 48 points if central column is otherwise empty — LPTS = 168
- 32 points for each front unit whose retreat is blocked — LPTS = 168
- 2^{Δ} points for each column pair, where Δ is the difference in LV (iff $\Delta > 0$)

Δ values for this example:

Lead Column \ Lag Column	1	2	3	4	5
1	–	<	<	<	0
2	4	–	1	2	4
3	3	<	–	1	3
4	2	<	<	–	2
5	0	<	<	<	–

Hence total penalty in this example is:
$2^4 + 2^1 + 2^2 + 2^4 + 2^3 + 2^1 + 2^3 + 2^2 + 2^2 = 64$

LPTS = 104 final

B. Accumulated Points [ACCLO, ACCHI]

$$ACC = \sum_{SECDIR=0}^{3} LPTS_{SECDIR} * IFRX_{SECDIR}$$

C. Computation of Square Value [SQVAL]

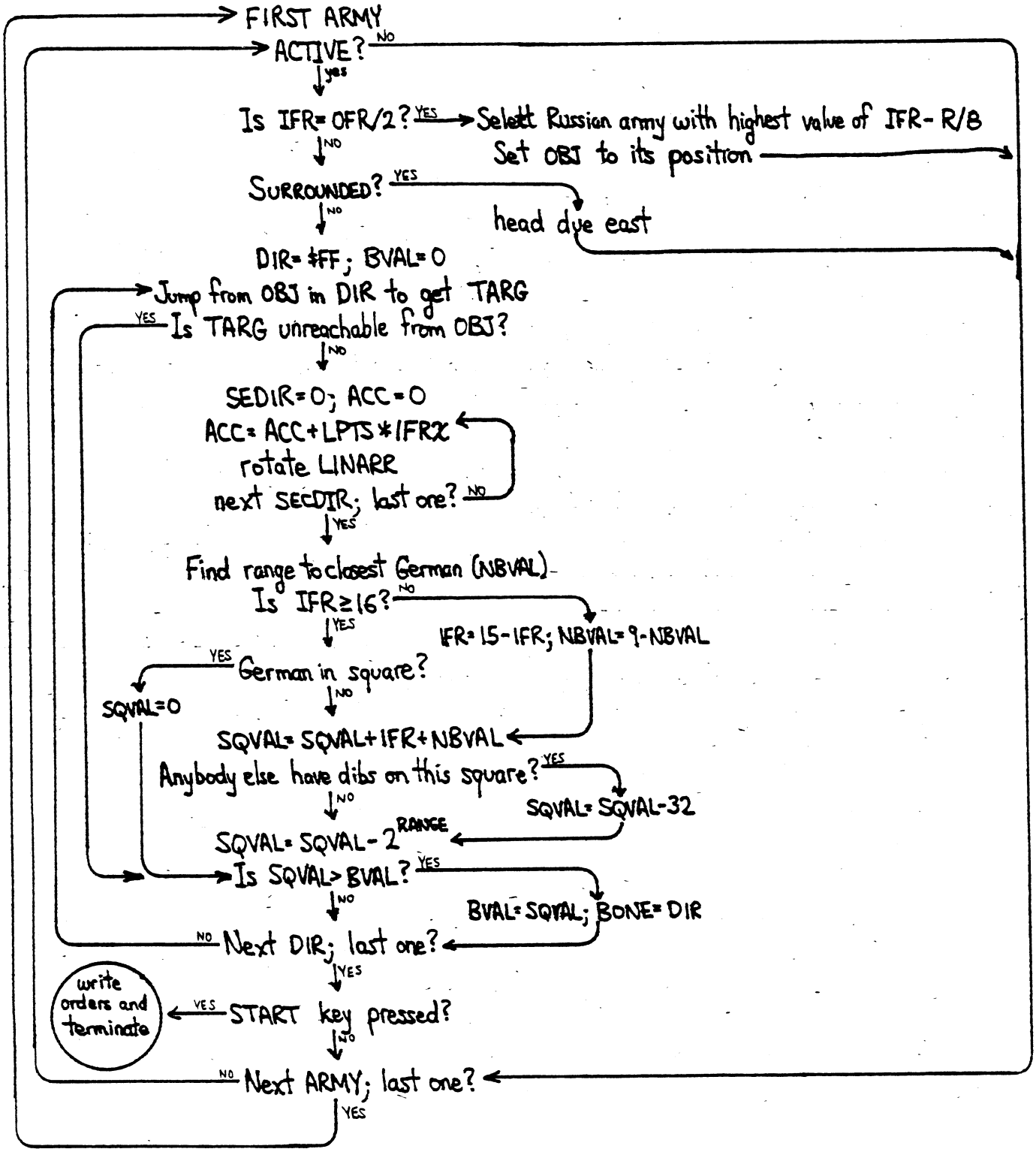
Start with SQVAL = ACCHI
Determine NBVAL, range to nearest German

If IFR≥16 (defensive strategy):
    If NBVAL=0 (i.e., German in square), then SQVAL=0, exit?
    Add IFR* (NBVAL+defensive bonus) to SQVAL
If IFR<15 (offensive strategy):
    Add 2*(15-IFR)* (9-NBVAL+defensive bonus) to SQVAL

If somebody else has dibs on this square, SQVAL= SQVAL-32

SQVAL= SQVAL- 2^R where R is range from unit to objective

TUMBLECHART FOR RUSSIAN MOVE (CENTRAL PORTION)



TERRAIN VALUES

TERRAIN TYPE	SUBTURN DELAY						DEFENSIVE VALUE	OFFENSIVE VALUE
	DRY		MUD		SNOW			
	Inf	Arm	Inf	Arm	Inf	Arm		
Clear	6	4	24	30	10	6	2	1
Mountain/Forest	12	8	30	30	16	10	3	1
City	18	6	24	30	10	8	3	1
Frozen Swamp	0	0	0	0	12	8	2	1
Frozen River	0	0	0	0	12	8	2	1
Swamp	18	18	30	30	24	24	2	1
River	14	13	30	30	28	28	1	2
Coastline	8	6	26	30	12	8	1	2
Estuary	20	16	28	30	24	20	2	1
Open Sea	127	127	127	127	127	127	0	0

```
                    10 ;EFT VERSION 1.8D (DATA) 11/30/81  COPYRIGHT CHRIS CRAWFORD 1981
0000                20          *=   $5400
5400 00             30 CORPSX   .BYTE 0,40,40,40,40,40,41,40,41,41,41
5401 28
5402 28
5403 28
5404 28
5405 28
5406 29
5407 28
5408 29
5409 29
540A 29
540B 2A             40          .BYTE 42,42,42,42,43,43,43,41,40,40,41,41
540C 2A
540D 2A
540E 2A
540F 2B
5410 2B
5411 2B
5412 29
5413 28
5414 28
5415 29
5416 29
5417 2A             50          .BYTE 42,42,42,40,41,42,41,42,42,43,41,42
5418 2A
5419 2A
541A 28
541B 29
541C 2A
541D 29
541E 2A
541F 2A
5420 2B
5421 29
5422 2A
5423 2B             60          .BYTE 43,30,30,31,33,35,37,35,36,36,45,45
5424 1E
5425 1E
5426 1F
5427 21
5428 23
5429 25
542A 23
542B 24
542C 24
542D 2D
542E 2D
542F 26             70          .BYTE 38,45,31,45,45,32,45,45
5430 2D
5431 1F
5432 2D
5433 2D
5434 20
5435 2D
5436 2D
                    80 ;RUSSIAN
5437 1D             90          .BYTE 29,27,24,23
5438 1B
5439 18
543A 17
543B 14             0100        .BYTE 20,15,0,0,0,0,0,0,0,0,0,0
543C 0F
543D 00
543E 00
543F 00
5440 00
5441 00
5442 00
5443 00
5444 00
5445 00
5446 00
5447 15             0110        .BYTE 21,21,30,30,39,38,23,19,34,34,31,27
5448 15
5449 1E
544A 1E
544B 27
544C 26
544D 17
544E 13
544F 22
5450 22
5451 1F
5452 1B
5453 21             0120        .BYTE 33,41,40,39,42,39,39,39,39,39,37,39
5454 29
5455 28
5456 27
5457 2A
5458 27
5459 27
545A 27
545B 27
545C 27
545D 25
545E 27
545F 27             0130        .BYTE 39,39,40,41,41,39,36,34,32,35,30,28
5460 27
5461 28
5462 29
5463 29
```

```
5464 27
5465 24
5466 22
5467 20
5468 23
5469 1E
546A 1C
546B 19     0140        .BYTE 25,29,32,33,26,21,29,0,28,21,21,21
546C 1D
546D 20
546E 21
546F 1A
5470 15
5471 1D
5472 00
5473 1C
5474 15
5475 15
5476 15
5477 14     0150        .BYTE 20,20,12,0,0,0,0,0,0,0,21,25
5478 14
5479 0C
547A 00
547B 00
547C 00
547D 00
547E 00
547F 00
5480 00
5481 15
5482 19
5483 00     0160        .BYTE 0,0,0,0,0,0,0,0,0,0,0,0,0,0
5484 00
5485 00
5486 00
5487 00
5488 00
5489 00
548A 00
548B 00
548C 00
548D 00
548E 00
548F 00
5490 00
5491 00     0170        .BYTE 0,0,0,0,38,21,12,20,21,20,15,21,20,19
5492 00
5493 00
5494 00
5495 26
5496 15
5497 0C
5498 14
5499 15
549A 14
549B 0F
549C 15
549D 14
549E 13
549F 00     0180 CORPSY .BYTE 0,20,19,18,17,16,20,19,18,17,16
54A0 14
54A1 13
54A2 12
54A3 11
54A4 10
54A5 14
54A6 13
54A7 12
54A8 11
54A9 10
54AA 14     0190        .BYTE 20,19,18,17,19,18,17,23,22,21,21,22
54AB 13
54AC 12
54AD 11
54AE 13
54AF 12
54B0 11
54B1 17
54B2 16
54B3 15
54B4 15
54B5 16
54B6 16     0200        .BYTE 22,23,24,15,14,13,15,14,12,13,15,16
54B7 17
54B8 18
54B9 0F
54BA 0E
54BB 0D
54BC 0F
54BD 0E
54BE 0C
54BF 0D
54C0 0F
54C1 10
54C2 10     0210        .BYTE 16,2,3,4,6,7,8,38,37,38,20,15,8
54C3 02
54C4 03
54C5 04
54C6 06
54C7 07
54C8 08
54C9 26
54CA 25
54CB 26
```

```
54CC 14
54CD 0F
54CE 08
54CF 10     0220        .BYTE 16,1,20,19,1,17,18
54D0 01
54D1 14
54D2 13
54D3 01
54D4 11
54D5 12
            0230 ;RUSSIAN
54D6 20     0240        .BYTE 32,31,38,38,38,38
54D7 1F
54D8 26
54D9 26
54DA 26
54DB 26
54DC 14     0250        .BYTE 20,8,18,10,14,33,11,15,20,10
54DD 08
54DE 12
54DF 0A
54E0 0E
54E1 21
54E2 0B
54E3 0F
54E4 14
54E5 0A
54E6 1C     0260        .BYTE 28,27,14,13,28,28,31,24,22,21,34,6
54E7 1B
54E8 0E
54E9 0D
54EA 1C
54EB 1C
54EC 1F
54ED 18
54EE 16
54EF 15
54F0 22
54F1 06
54F2 25     0270        .BYTE 37,24,23,23,25,20,22,18,17,21,20,19
54F3 18
54F4 17
54F5 17
54F6 19
54F7 14
54F8 16
54F9 12
54FA 11
54FB 15
54FC 14
54FD 13
54FE 10     0280        .BYTE 16,15,14,13,12,11,9,8,6,9,4,2,6
54FF 0F
5500 0E
5501 0D
5502 0C
5503 0B
5504 09
5505 08
5506 06
5507 09
5508 04
5509 02
550A 06
550B 0E     0290        .BYTE 14,22,36,23,8,33,28,30,20,28,33,27
550C 16
550D 24
550E 17
550F 08
5510 21
5511 1C
5512 1E
5513 14
5514 1C
5515 21
5516 1B
5517 1E     0300        .BYTE 30,8,10,32,11,25,12,23,13,29,30,31
5518 08
5519 0A
551A 20
551B 0B
551C 19
551D 0C
551E 17
551F 0D
5520 1D
5521 1E
5522 1F
5523 0F     0310        .BYTE 15,27,17,25,11,23,19,21,33,28,13,26
5524 1B
5525 11
5526 19
5527 0B
5528 17
5529 13
552A 15
552B 21
552C 1C
552D 0D
552E 1A
552F 0A     0320        .BYTE 10,29,35,27,15,30,22,8,13,14,28
5530 1D
5531 23
5532 1B
```

```
5533 0F
5534 1E
5535 16
5536 08
5537 0D
5538 0E
5539 1C
553A 03     0330        .BYTE 3,3,3,2
553B 03
553C 03
553D 02
553E 00     0340 MSTRNG .BYTE 0,203,205,192,199,184,136,127,150
553F CB
5540 CD
5541 C0
5542 C7
5543 B8
5544 88
5545 7F
5546 96
5547 81     0350        .BYTE 129,136,109,72,70,81,131,102,53
5548 88
5549 6D
554A 48
554B 46
554C 51
554D 83
554E 66
554F 35
5550 C6     0360        .BYTE 198,194,129,123,101,104,112,120
5551 C2
5552 81
5553 7B
5554 65
5555 68
5556 70
5557 78
5558 CA     0370        .BYTE 202,195,191,72,140,142,119,111
5559 C3
555A BF
555B 48
555C 8C
555D 8E
555E 77
555F 6F
5560 7A     0380        .BYTE 122,77,97,96,92,125,131,106
5561 4D
5562 61
5563 60
5564 5C
5565 7D
5566 83
5567 6A
5568 70     0390        .BYTE 112,104,101,210,97,98,95,52
5569 68
556A 65
556B D2
556C 61
556D 62
556E 5F
556F 34
5570 62     0400        .BYTE 98,96,55,104,101
5571 60
5572 37
5573 68
5574 65
            0410 ;RUSSIAN
5575 64     0420        .BYTE 100,103,110
5576 67
5577 6E
5578 65     0430        .BYTE 101,92,103,105,107,111,88,117,84
5579 5C
557A 67
557B 69
557C 6B
557D 6F
557E 58
557F 75
5580 54
5581 6D     0440        .BYTE 109,89,105,93
5582 59
5583 69
5584 5D
5585 3E     0450        .BYTE 62,104,101,67,104,84,127,112
5586 68
5587 65
5588 43
5589 68
558A 54
558B 7F
558C 70
558D 6F     0460        .BYTE 111,91,79,69,108,118,137,70
558E 5B
558F 4F
5590 45
5591 6C
5592 76
5593 89
5594 46
5595 55     0470        .BYTE 85,130,91,131,71,86,75,90
5596 82
5597 5B
5598 83
5599 47
```

```
559A 56
559B 4B
559C 5A
559D 7B  0480        .BYTE 123,124,151,126,88,77,79,80
559E 7C
559F 97
55A0 80
55A1 58
55A2 4D
55A3 4F
55A4 50
55A5 7E  0490        .BYTE 126,79,91,84,72,86,76,99
55A6 4F
55A7 5B
55A8 54
55A9 48
55AA 56
55AB 4C
55AC 63
55AD 43  0500        .BYTE 67,78,121,114,105,122,127,129
55AE 4E
55AF 79
55B0 72
55B1 69
55B2 7A
55B3 7F
55B4 81
55B5 69  0510        .BYTE 105,111,112,127,119,89,108
55B6 6F
55B7 70
55B8 7F
55B9 77
55BA 59
55BB 6C
55BC 71  0520        .BYTE 113,105,94,103,97,108,110,111
55BD 69
55BE 5E
55BF 67
55C0 61
55C1 6C
55C2 6E
55C3 6F
55C4 60  0530        .BYTE 96,109,112,95,93,114,103,107
55C5 6D
55C6 70
55C7 5F
55C8 5D
55C9 72
55CA 67
55CB 6B
55CC 69  0540        .BYTE 105,92,109,101,106,95,99,101
55CD 5C
55CE 6D
55CF 65
55D0 6A
55D1 5F
55D2 63
55D3 65
55D4 76  0550        .BYTE 118,106,112,104,185,108,94,102
55D5 6A
55D6 70
55D7 68
55D8 B9
55D9 6C
55DA 5E
55DB 66
55DC 62  0560        .BYTE 98
55DD     0570 CSTRNG *=   *+159
567C 00  0580 SWAP   .BYTE 0,126,126,126,126,126,125,125,125
567D 7E
567E 7E
567F 7E
5680 7E
5681 7E
5682 7D
5683 7D
5684 7D
5685 7D  0590        .BYTE 125,125,125,125,125,125,125,125
5686 7D
5687 7D
5688 7D
5689 7D
568A 7D
568B 7D
568C 7D
568D 7D  0600        .BYTE 125,126,126,125,125,125,125,125
568E 7E
568F 7E
5690 7D
5691 7D
5692 7D
5693 7D
5694 7D
5695 7D  0610        .BYTE 125,126,126,126,126,125,125,125
5696 7E
5697 7E
5698 7E
5699 7E
569A 7D
569B 7D
569C 7D
569D 7D  0620        .BYTE 125,125,125,125,125,125,125,125
569E 7D
569F 7D
```

```
56A0 7D
56A1 7D
56A2 7D
56A3 7D
56A4 7D
56A5 7D     0630        .BYTE 125,125,125,125,126,125,126,125
56A6 7D
56A7 7D
56A8 7D
56A9 7E
56AA 7D
56AB 7E
56AC 7D
56AD 7D     0640        .BYTE 125,125,125,125,125,126
56AE 7D
56AF 7D
56B0 7D
56B1 7D
56B2 7E
            0650 ;RUSSIAN
56B3 FD     0660        .BYTE 253,253
56B4 FD
56B5 FD     0670        .BYTE 253,253,253,253,253,253,253,253
56B6 FD
56B7 FD
56B8 FD
56B9 FD
56BA FD
56BB FD
56BC FD
56BD FE     0680        .BYTE 254,254,254,254,254,254
56BE FE
56BF FE
56C0 FE
56C1 FE
56C2 FE
56C3 FE     0690        .BYTE 254,253,253,254,253,254,253,253
56C4 FD
56C5 FD
56C6 FE
56C7 FD
56C8 FE
56C9 FD
56CA FD
56CB FD     0700        .BYTE 253,254,253,253,253,253,253,254
56CC FE
56CD FD
56CE FD
56CF FD
56D0 FD
56D1 FD
56D2 FE
56D3 FE     0710        .BYTE 254,253,253,253,254,254,254,254
56D4 FD
56D5 FD
56D6 FD
56D7 FE
56D8 FE
56D9 FE
56DA FE
56DB FD     0720        .BYTE 253,253,253,253,254,254,254,254
56DC FD
56DD FD
56DE FD
56DF FE
56E0 FE
56E1 FE
56E2 FE
56E3 FD     0730        .BYTE 253,254,254,254,253,253,253,253
56E4 FE
56E5 FE
56E6 FE
56E7 FD
56E8 FD
56E9 FD
56EA FD
56EB FD     0740        .BYTE 253,253,253,253,253,253,253,253
56EC FD
56ED FD
56EE FD
56EF FD
56F0 FD
56F1 FD
56F2 FD
56F3 FD     0750        .BYTE 253,253,253,253,253,253,253,253
56F4 FD
56F5 FD
56F6 FD
56F7 FD
56F8 FD
56F9 FD
56FA FD
56FB FD     0760        .BYTE 253,253,254,253,253,253,253,253
56FC FD
56FD FE
56FE FD
56FF FD
5700 FD
5701 FD
5702 FD
5703 FD     0770        .BYTE 253,253,253,253,254,254,254,253
5704 FD
5705 FD
5706 FD
```

```
5707 FE
5708 FE
5709 FE
570A FD
570B FE     0780        .BYTE 254,253,254,254,253,254,253,253
570C FD
570D FE
570E FE
570F FD
5710 FE
5711 FD
5712 FD
5713 FD     0790        .BYTE 253,253,253,253,253,253,253,253
5714 FD
5715 FD
5716 FD
5717 FD
5718 FD
5719 FD
571A FD
571B FF     0800 ARRIVE .BYTE 255,0,255,0,0,0,0,0,0
571C 00
571D FF
571E 00
571F 00
5720 00
5721 00
5722 00
5723 00
5724 00     0810        .BYTE 0,0,255,255,255,255,255,255
5725 00
5726 FF
5727 FF
5728 FF
5729 FF
572A FF
572B FF
572C FF     0820        .BYTE 255,0,0,0,0,0,0,0
572D 00
572E 00
572F 00
5730 00
5731 00
5732 00
5733 00
5734 00     0830        .BYTE 0,0,0,0,255,0,0,0
5735 00
5736 00
5737 00
5738 FF
5739 00
573A 00
573B 00
573C 00     0840        .BYTE 0,255,255,0,0,0,0,0
573D FF
573E FF
573F 00
5740 00
5741 00
5742 00
5743 00
5744 00     0850        .BYTE 0,0,0,255,2,255,2
5745 00
5746 00
5747 FF
5748 02
5749 FF
574A 02
574B 05     0860        .BYTE 5,6,9,10,11,20,24
574C 06
574D 09
574E 0A
574F 0B
5750 14
5751 18
            0870 ;RUSSIAN
5752 04     0880        .BYTE 4,5,7,9,11,13,7,12,8
5753 05
5754 07
5755 09
5756 0B
5757 0D
5758 07
5759 0C
575A 08
575B 0A     0890        .BYTE 10,10,14,15,16,18,7
575C 0A
575D 0E
575E 0F
575F 10
5760 12
5761 07
5762 00     0900        .BYTE 0,0,0,0,0,0,0,0
5763 00
5764 00
5765 00
5766 00
5767 00
5768 00
5769 00
576A 00     0910        .BYTE 0,0,0,0,0,0,0,0
576B 00
576C 00
576D 00
```

```
576E 00
576F 00
5770 00
5771 00
5772 00     0920        .BYTE 0,0,0,0,0,0,0,0
5773 00
5774 00
5775 00
5776 00
5777 00
5778 00
5779 00
577A 00     0930        .BYTE 0,0,0,0,0,0,0,0
577B 00
577C 00
577D 00
577E 00
577F 00
5780 00
5781 00
5782 00     0940        .BYTE 0,0,0,0,1,1,1,1
5783 00
5784 00
5785 00
5786 01
5787 01
5788 01
5789 01
578A 01     0950        .BYTE 1,2,2,2,3,3,4,4
578B 02
578C 02
578D 02
578E 03
578F 03
5790 04
5791 04
5792 05     0960        .BYTE 5,5,6,6,7,8,8,8
5793 05
5794 06
5795 06
5796 07
5797 08
5798 08
5799 08
579A 09     0970        .BYTE 9,9,5,5,2,9,10,10
579B 09
579C 05
579D 05
579E 02
579F 09
57A0 0A
57A1 0A
57A2 06     0980        .BYTE 6,11,5,17,2,11,20,21
57A3 0B
57A4 05
57A5 11
57A6 02
57A7 0B
57A8 14
57A9 15
57AA 16     0990        .BYTE 22,23,24,26,28,30,2,3
57AB 17
57AC 18
57AD 1A
57AE 1C
57AF 1E
57B0 02
57B1 03
57B2 03     1000        .BYTE 3,3,3,6,6,4,4,4
57B3 03
57B4 03
57B5 06
57B6 06
57B7 04
57B8 04
57B9 04
57BA 20     1010 WORDS  .BYTE "        SS       "
57BB 20
57BC 20
57BD 20
57BE 20
57BF 20
57C0 20
57C1 20
57C2 53
57C3 53
57C4 20
57C5 20
57C6 20
57C7 20
57C8 20
57C9 20
57CA 46     1020        .BYTE "FINNISH RUMANIAN"
57CB 49
57CC 4E
57CD 4E
57CE 49
57CF 53
57D0 48
57D1 20
57D2 52
57D3 55
57D4 4D
57D5 41
```

```
57D6 4E
57D7 49
57D8 41
57D9 4E
57DA 49     1030        .BYTE "ITALIAN HUNGARAN"
57DB 54
57DC 41
57DD 4C
57DE 49
57DF 41
57E0 4E
57E1 20
57E2 48
57E3 55
57E4 4E
57E5 47
57E6 41
57E7 52
57E8 41
57E9 4E
57EA 4D     1040        .BYTE "MOUNTAINGUARDS  "
57EB 4F
57EC 55
57ED 4E
57EE 54
57EF 41
57F0 49
57F1 4E
57F2 47
57F3 55
57F4 41
57F5 52
57F6 44
57F7 53
57F8 20
57F9 20
57FA 49     1050        .BYTE "INFANTRYTANK    "
57FB 4E
57FC 46
57FD 41
57FE 4E
57FF 54
5800 52
5801 59
5802 54
5803 41
5804 4E
5805 4B
5806 20
5807 20
5808 20
5809 20
580A 43     1060        .BYTE "CAVALRY PANZER  "
580B 41
580C 56
580D 41
580E 4C
580F 52
5810 59
5811 20
5812 50
5813 41
5814 4E
5815 5A
5816 45
5817 52
5818 20
5819 20
581A 4D     1070        .BYTE "MILITIA SHOCK   "
581B 49
581C 4C
581D 49
581E 54
581F 49
5820 41
5821 20
5822 53
5823 48
5824 4F
5825 43
5826 4B
5827 20
5828 20
5829 20
582A 50     1080        .BYTE "PARATRP PZRGRNDR"
582B 41
582C 52
582D 41
582E 54
582F 52
5830 50
5831 20
5832 50
5833 5A
5834 52
5835 47
5836 52
5837 4E
5838 44
5839 52
583A 20     1090        .BYTE "        JANUARY "
583B 20
583C 20
583D 20
```

```
583E 20
583F 20
5840 20
5841 20
5842 4A
5843 41
5844 4E
5845 55
5846 41
5847 52
5848 59
5849 20
584A 46     1100        .BYTE "FEBRUARYMARCH   "
584B 45
584C 42
584D 52
584E 55
584F 41
5850 52
5851 59
5852 4D
5853 41
5854 52
5855 43
5856 48
5857 20
5858 20
5859 20
585A 41     1110        .BYTE "APRIL   MAY     "
585B 50
585C 52
585D 49
585E 4C
585F 20
5860 20
5861 20
5862 4D
5863 41
5864 59
5865 20
5866 20
5867 20
5868 20
5869 20
586A 4A     1120        .BYTE "JUNE    JULY    "
586B 55
586C 4E
586D 45
586E 20
586F 20
5870 20
5871 20
5872 4A
5873 55
5874 4C
5875 59
5876 20
5877 20
5878 20
5879 20
587A 41     1130        .BYTE "AUGUST  SEPTEMBR"
587B 55
587C 47
587D 55
587E 53
587F 54
5880 20
5881 20
5882 53
5883 45
5884 50
5885 54
5886 45
5887 4D
5888 42
5889 52
588A 4F     1140        .BYTE "OCTOBER NOVEMBER"
588B 43
588C 54
588D 4F
588E 42
588F 45
5890 52
5891 20
5892 4E
5893 4F
5894 56
5895 45
5896 4D
5897 42
5898 45
5899 52
589A 44     1150        .BYTE "DECEMBERCORPS   "
589B 45
589C 43
589D 45
589E 4D
589F 42
58A0 45
58A1 52
58A2 43
58A3 4F
58A4 52
58A5 50
```

```
58A6 53
58A7 20
58A8 20
58A9 20
58AA 41     1160        .BYTE "ARMY    MUSTER  "
58AB 52
58AC 4D
58AD 59
58AE 20
58AF 20
58B0 20
58B1 20
58B2 4D
58B3 55
58B4 53
58B5 54
58B6 45
58B7 52
58B8 20
58B9 20
58BA 43     1170        .BYTE "COMBAT  STRENGTH"
58BB 4F
58BC 4D
58BD 42
58BE 41
58BF 54
58C0 20
58C1 20
58C2 53
58C3 54
58C4 52
58C5 45
58C6 4E
58C7 47
58C8 54
58C9 48
58CA 00     1180 CORPT  .BYTE 0,3,3,3,3,3,0,0
58CB 03
58CC 03
58CD 03
58CE 03
58CF 03
58D0 00
58D1 00
58D2 00     1190        .BYTE 0,0,0,0,0,0,0,0
58D3 00
58D4 00
58D5 00
58D6 00
58D7 00
58D8 00
58D9 00
58DA 00     1200        .BYTE 0,$40,3,3,0,0,0,0
58DB 40
58DC 03
58DD 03
58DE 00
58DF 00
58E0 00
58E1 00
58E2 00     1210        .BYTE 0,0,3,3,3,3,0,0
58E3 00
58E4 03
58E5 03
58E6 03
58E7 03
58E8 00
58E9 00
58EA 00     1220        .BYTE 0,0,0,0,$30,$30,$30,0
58EB 00
58EC 00
58ED 00
58EE 30
58EF 30
58F0 30
58F1 00
58F2 00     1230        .BYTE 0,0,$20,$20,$20,3,0,$53
58F3 00
58F4 20
58F5 20
58F6 20
58F7 03
58F8 00
58F9 53
58FA 00     1240        .BYTE 0,$30,0,0,$40,0,7
58FB 30
58FC 00
58FD 00
58FE 40
58FF 00
5900 07
            1250 ;RUSSIAN
5901 04     1260        .BYTE 4,4,0,0,0,0,0,0,0
5902 04
5903 00
5904 00
5905 00
5906 00
5907 00
5908 00
5909 00
590A 00     1270        .BYTE 0,1,1,1,1,1,2
590B 01
590C 01
```

```
590D 01
590E 01
590F 01
5910 02
5911 01     1280        .BYTE 1,0,0,2,0,1,0,0
5912 00
5913 00
5914 02
5915 00
5916 01
5917 00
5918 00
5919 00     1290        .BYTE 0,1,4,0,4,0,0,1
591A 01
591B 04
591C 00
591D 04
591E 00
591F 00
5920 01
5921 01     1300        .BYTE 1,0,0,0,1,1,2,2
5922 00
5923 00
5924 00
5925 01
5926 01
5927 02
5928 02
5929 00     1310        .BYTE 0,0,0,0,1,1,1,2
592A 00
592B 00
592C 00
592D 01
592E 01
592F 01
5930 02
5931 00     1320        .BYTE 0,1,2,2,0,4,0,4
5932 01
5933 02
5934 02
5935 00
5936 04
5937 00
5938 04
5939 00     1330        .BYTE 0,0,0,0,0,0,0,0
593A 00
593B 00
593C 00
593D 00
593E 00
593F 00
5940 00
5941 00     1340        .BYTE 0,0,0,0,0,0,0,0
5942 00
5943 00
5944 00
5945 00
5946 00
5947 00
5948 00
5949 00     1350        .BYTE 0,0,$72,0,$70,$70,$70,$70
594A 00
594B 72
594C 00
594D 70
594E f>g4F 70
5950 70
5951 00     1360        .BYTE 0,0,0,0,$72,1,$71,$70
5952 00
5953 00
5954 00
5955 72
5956 01
5957 71
5958 70
5959 01     1370        .BYTE 1,$70,1,1,0,0,0,0
595A 70
595B 01
595C 01
595D 00
595E 00
595F 00
5960 00
5961 00     1380        .BYTE 0,0,0,4,4,4,4,4
5962 00
5963 00
5964 04
5965 04
5966 04
5967 04
5968 04
5969 00     1390 CORPNO .BYTE 0,24,39,46,47,57,5,6
596A 18
596B 27
596C 2E
596D 2F
596E 39
596F 05
5970 06
5971 07     1400        .BYTE 7,8,9,12,13,20,42,43
5972 08
5973 09
5974 0C
```

```
5975 0D
5976 14
5977 2A
5978 2B
5979 35     1410        .BYTE 53,3,41,56,1,2,10,26
597A 03
597B 29
597C 38
597D 01
597E 02
597F 0A
5980 1A
5981 1C     1420        .BYTE 28,38,3,14,48,52,49,4
5982 26
5983 03
5984 0E
5985 30
5986 34
5987 31
5988 04
5989 11     1430        .BYTE 17,29,44,55,1,2,4,11
598A 1D
598B 2C
598C 37
598D 01
598E 02
598F 04
5990 0B
5991 1E     1440        .BYTE 30,54,2,4,6,40,27,1
5992 36
5993 02
5994 04
5995 06
5996 28
5997 1B
5998 01
5999 17     1450        .BYTE 23,5,34,35,4,51,50
599A 05
599B 22
599C 23
599D 04
599E 33
599F 32
            1460 ;RUSSIAN
59A0 07     1470        .BYTE 7,11,41,42,43,44,45,46
59A1 0B
59A2 29
59A3 2A
59A4 2B
59A5 2C
59A6 2D
59A7 2E
59A8 2F     1480        .BYTE 47,48,9,13,14,15,16,7
59A9 30
59AA 09
59AB 0D
59AC 0E
59AD 0F
59AE 10
59AF 07
59B0 02     1490        .BYTE 2,19,18,1,27,10,22,21
59B1 13
59B2 12
59B3 01
59B4 1B
59B5 0A
59B6 16
59B7 15
59B8 0D     1500        .BYTE 13,6,9,2,1,8,11,1
59B9 06
59BA 09
59BB 02
59BC 01
59BD 08
59BE 0B
59BF 01
59C0 07     1510        .BYTE 7,3,4,10,5,8,3,6
59C1 03
59C2 04
59C3 0A
59C4 05
59C5 08
59C6 03
59C7 06
59C8 05     1520        .BYTE 5,6,12,26,3,4,11,5
59C9 06
59CA 0C
59CB 1A
59CC 03
59CD 04
59CE 0B
59CF 05
59D0 09     1530        .BYTE 9,12,4,2,7,2,14,4
59D1 0C
59D2 04
59D3 02
59D4 07
59D5 02
59D6 0E
59D7 04
59D8 0F     1540        .BYTE 15,16,20,6,24,40,29,30
59D9 10
59DA 14
59DB 06
```

```
59DC 18
59DD 28
59DE 1D
59DF 1E
59E0 1F     1550        .BYTE 31,32,33,37,43,49,50,52
59E1 20
59E2 21
59E3 25
59E4 2B
59E5 31
59E6 32
59E7 34
59E8 36     1560        .BYTE 54,55,1,34,1,2,3,4
59E9 37
59EA 01
59EB 22
59EC 01
59ED 02
59EE 03
59EF 04
59F0 27     1570        .BYTE 39,59,60,61,2,1,1,5
59F1 3B
59F2 3C
59F3 3D
59F4 02
59F5 01
59F6 01
59F7 05
59F8 02     1580        .BYTE 2,6,3,4,38,36,35,28
59F9 06
59FA 03
59FB 04
59FC 26
59FD 24
59FE 23
59FF 1C
5A00 19     1590        .BYTE 25,23,17,8,10,3,5,6
5A01 17
5A02 11
5A03 08
5A04 0A
5A05 03
5A06 05
5A07 06
            1600 ;HERE COME NUMBER CODES
5A08 00     1610 HDIGIT .BYTE 0,0,0,0,0,0,0,0
5A09 00
5A0A 00
5A0B 00
5A0C 00
5A0D 00
5A0E 00
5A0F 00
5A10 00     1620        .BYTE 0,0,0,0,0,0,0,0
5A11 00
5A12 00
5A13 00
5A14 00
5A15 00
5A16 00
5A17 00
5A18 00     1630        .BYTE 0,0,0,0,0,0,0,0
5A19 00
5A1A 00
5A1B 00
5A1C 00
5A1D 00
5A1E 00
5A1F 00
5A20 00     1640        .BYTE 0,0,0,0,0,0,0,0
5A21 00
5A22 00
5A23 00
5A24 00
5A25 00
5A26 00
5A27 00
5A28 00     1650        .BYTE 0,0,0,0,0,0,0,0
5A29 00
5A2A 00
5A2B 00
5A2C 00
5A2D 00
5A2E 00
5A2F 00
5A30 00     1660        .BYTE 0,0,0,0,0,0,0,0
5A31 00
5A32 00
5A33 00
5A34 00
5A35 00
5A36 00
5A37 00
5A38 00     1670        .BYTE 0,0,0,0,0,0,0,0
5A39 00
5A3A 00
5A3B 00
5A3C 00
5A3D 00
5A3E 00
5A3F 00
5A40 00     1680        .BYTE 0,0,0,0,0,0,0,0
5A41 00
5A42 00
```

```
5A43 00
5A44 00
5A45 00
5A46 00
5A47 00
5A48 00     1690        .BYTE 0,0,0,0,0,0,0,0
5A49 00
5A4A 00
5A4B 00
5A4C 00
5A4D 00
5A4E 00
5A4F 00
5A50 00     1700        .BYTE 0,0,0,0,0,0,0,0
5A51 00
5A52 00
5A53 00
5A54 00
5A55 00
5A56 00
5A57 00
5A58 00     1710        .BYTE 0,0,0,0,0,0,0,0
5A59 00
5A5A 00
5A5B 00
5A5C 00
5A5D 00
5A5E 00
5A5F 00
5A60 00     1720        .BYTE 0,0,0,0,0,0,0,0
5A61 00
5A62 00
5A63 00
5A64 00
5A65 00
5A66 00
5A67 00
5A68 00     1730        .BYTE 0,0,0,0,1,1,1,1
5A69 00
5A6A 00
5A6B 00
5A6C 01
5A6D 01
5A6E 01
5A6F 01
5A70 01     1740        .BYTE 1,1,1,1,1,1,1,1
5A71 01
5A72 01
5A73 01
5A74 01
5A75 01
5A76 01
5A77 01
5A78 01     1750        .BYTE 1,1,1,1,1,1,1,1
5A79 01
5A7A 01
5A7B 01
5A7C 01
5A7D 01
5A7E 01
5A7F 01
5A80 01     1760        .BYTE 1,1,1,1,1,1,1,1
5A81 01
5A82 01
5A83 01
5A84 01
5A85 01
5A86 01
5A87 01
5A88 01     1770        .BYTE 1,1,1,1,1,1,1,1
5A89 01
5A8A 01
5A8B 01
5A8C 01
5A8D 01
5A8E 01
5A8F 01
5A90 01     1780        .BYTE 1,1,1,1,1,1,1,1
5A91 01
5A92 01
5A93 01
5A94 01
5A95 01
5A96 01
5A97 01
5A98 01     1790        .BYTE 1,1,1,1,1,1,1,1
5A99 01
5A9A 01
5A9B 01
5A9C 01
5A9D 01
5A9E 01
5A9F 01
5AA0 01     1800        .BYTE 1,1,1,1,1,1,1,1
5AA1 01
5AA2 01
5AA3 01
5AA4 01
5AA5 01
5AA6 01
5AA7 01
5AA8 01     1810        .BYTE 1,1,1,1,1,1,1,1
5AA9 01
5AAA 01
```

```
5AAB 01
5AAC 01
5AAD 01
5AAE 01
5AAF 01
5AB0 01      1820        .BYTE 1,1,1,1,1,1,1,1
5AB1 01
5AB2 01
5AB3 01
5AB4 01
5AB5 01
5AB6 01
5AB7 01
5AB8 01      1830        .BYTE 1,1,1,1,1,1,1,1
5AB9 01
5ABA 01
5ABB 01
5ABC 01
5ABD 01
5ABE 01
5ABF 01
5AC0 01      1840        .BYTE 1,1,1,1,1,1,1,1
5AC1 01
5AC2 01
5AC3 01
5AC4 01
5AC5 01
5AC6 01
5AC7 01
5AC8 01      1850        .BYTE 1,1,1,1,1,1,1,1
5AC9 01
5ACA 01
5ACB 01
5ACC 01
5ACD 01
5ACE 01
5ACF 01
5AD0 02      1860        .BYTE 2,2,2,2,2,2,2,2
5AD1 02
5AD2 02
5AD3 02
5AD4 02
5AD5 02
5AD6 02
5AD7 02
5AD8 02      1870        .BYTE 2,2,2,2,2,2,2,2
5AD9 02
5ADA 02
5ADB 02
5ADC 02
5ADD 02
5ADE 02
5ADF 02
5AE0 02      1880        .BYTE 2,2,2,2,2,2,2,2
5AE1 02
5AE2 02
5AE3 02
5AE4 02
5AE5 02
5AE6 02
5AE7 02
5AE8 02      1890        .BYTE 2,2,2,2,2,2,2,2
5AE9 02
5AEA 02
5AEB 02
5AEC 02
5AED 02
5AEE 02
5AEF 02
5AF0 02      1900        .BYTE 2,2,2,2,2,2,2,2
5AF1 02
5AF2 02
5AF3 02
5AF4 02
5AF5 02
5AF6 02
5AF7 02
5AF8 02      1910        .BYTE 2,2,2,2,2,2,2,2
5AF9 02
5AFA 02
5AFB 02
5AFC 02
5AFD 02
5AFE 02
5AFF 02
5B00 02      1920        .BYTE 2,2,2,2,2,2,2,2
5B01 02
5B02 02
5B03 02
5B04 02
5B05 02
5B06 02
5B07 02
5B08 00      1930 TDIGIT .BYTE 0,0,0,0,0,0,0,0,0,0
5B09 00
5B0A 00
5B0B 00
5B0C 00
5B0D 00
5B0E 00
5B0F 00
5B10 00
5B11 00
5B12 01      1940        .BYTE 1,1,1,1,1,1,1,1,1,1
```

```
5B13 01
5B14 01
5B15 01
5B16 01
5B17 01
5B18 01
5B19 01
5B1A 01
5B1B 01
5B1C 02     1950        .BYTE 2,2,2,2,2,2,2,2,2,2
5B1D 02
5B1E 02
5B1F 02
5B20 02
5B21 02
5B22 02
5B23 02
5B24 02
5B25 02
5B26 03     1960        .BYTE 3,3,3,3,3,3,3,3,3,3
5B27 03
5B28 03
5B29 03
5B2A 03
5B2B 03
5B2C 03
5B2D 03
5B2E 03
5B2F 03
5B30 04     1970        .BYTE 4,4,4,4,4,4,4,4,4,4
5B31 04
5B32 04
5B33 04
5B34 04
5B35 04
5B36 04
5B37 04
5B38 04
5B39 04
5B3A 05     1980        .BYTE 5,5,5,5,5,5,5,5,5,5
5B3B 05
5B3C 05
5B3D 05
5B3E 05
5B3F 05
5B40 05
5B41 05
5B42 05
5B43 05
5B44 06     1990        .BYTE 6,6,6,6,6,6,6,6,6,6
5B45 06
5B46 06
5B47 06
5B48 06
5B49 06
5B4A 06
5B4B 06
5B4C 06
5B4D 06
5B4E 07     2000        .BYTE 7,7,7,7,7,7,7,7,7,7
5B4F 07
5B50 07
5B51 07
5B52 07
5B53 07
5B54 07
5B55 07
5B56 07
5B57 07
5B58 08     2010        .BYTE 8,8,8,8,8,8,8,8,8,8
5B59 08
5B5A 08
5B5B 08
5B5C 08
5B5D 08
5B5E 08
5B5F 08
5B60 08
5B61 08
5B62 09     2020        .BYTE 9,9,9,9,9,9,9,9,9,9
5B63 09
5B64 09
5B65 09
5B66 09
5B67 09
5B68 09
5B69 09
5B6A 09
5B6B 09
5B6C 00     2030        .BYTE 0,0,0,0,0,0,0,0,0,0
5B6D 00
5B6E 00
5B6F 00
5B70 00
5B71 00
5B72 00
5B73 00
5B74 00
5B75 00
5B76 01     2040        .BYTE 1,1,1,1,1,1,1,1,1,1
5B77 01
5B78 01
5B79 01
5B7A 01
```

```
5B7B 01
5B7C 01
5B7D 01
5B7E 01
5B7F 01
5B80 02     2050        .BYTE 2,2,2,2,2,2,2,2,2,2
5B81 02
5B82 02
5B83 02
5B84 02
5B85 02
5B86 02
5B87 02
5B88 02
5B89 02
5B8A 03     2060        .BYTE 3,3,3,3,3,3,3,3,3,3
5B8B 03
5B8C 03
5B8D 03
5B8E 03
5B8F 03
5B90 03
5B91 03
5B92 03
5B93 03
5B94 04     2070        .BYTE 4,4,4,4,4,4,4,4,4,4
5B95 04
5B96 04
5B97 04
5B98 04
5B99 04
5B9A 04
5B9B 04
5B9C 04
5B9D 04
5B9E 05     2080        .BYTE 5,5,5,5,5,5,5,5,5,5
5B9F 05
5BA0 05
5BA1 05
5BA2 05
5BA3 05
5BA4 05
5BA5 05
5BA6 05
5BA7 05
5BA8 06     2090        .BYTE 6,6,6,6,6,6,6,6,6,6
5BA9 06
5BAA 06
5BAB 06
5BAC 06
5BAD 06
5BAE 06
5BAF 06
5BB0 06
5BB1 06
5BB2 07     2100        .BYTE 7,7,7,7,7,7,7,7,7,7
5BB3 07
5BB4 07
5BB5 07
5BB6 07
5BB7 07
5BB8 07
5BB9 07
5BBA 07
5BBB 07
5BBC 08     2110        .BYTE 8,8,8,8,8,8,8,8,8,8
5BBD 08
5BBE 08
5BBF 08
5BC0 08
5BC1 08
5BC2 08
5BC3 08
5BC4 08
5BC5 08
5BC6 09     2120        .BYTE 9,9,9,9,9,9,9,9,9,9
5BC7 09
5BC8 09
5BC9 09
5BCA 09
5BCB 09
5BCC 09
5BCD 09
5BCE 09
5BCF 09
5BD0 00     2130        .BYTE 0,0,0,0,0,0,0,0,0,0
5BD1 00
5BD2 00
5BD3 00
5BD4 00
5BD5 00
5BD6 00
5BD7 00
5BD8 00
5BD9 00
5BDA 01     2140        .BYTE 1,1,1,1,1,1,1,1,1,1
5BDB 01
5BDC 01
5BDD 01
5BDE 01
5BDF 01
5BE0 01
5BE1 01
5BE2 01
```

```
5BE3 01
5BE4 02     2150          .BYTE 2,2,2,2,2,2,2,2,2,2
5BE5 02
5BE6 02
5BE7 02
5BE8 02
5BE9 02
5BEA 02
5BEB 02
5BEC 02
5BED 02
5BEE 03     2160          .BYTE 3,3,3,3,3,3,3,3,3,3
5BEF 03
5BF0 03
5BF1 03
5BF2 03
5BF3 03
5BF4 03
5BF5 03
5BF6 03
5BF7 03
5BF8 04     2170          .BYTE 4,4,4,4,4,4,4,4,4,4
5BF9 04
5BFA 04
5BFB 04
5BFC 04
5BFD 04
5BFE 04
5BFF 04
5C00 04
5C01 04
5C02 05     2180          .BYTE 5,5,5,5,5,5
5C03 05
5C04 05
5C05 05
5C06 05
5C07 05
5C08 00     2190 ODIGIT   .BYTE 0,1,2,3,4,5,6,7,8,9
5C09 01
5C0A 02
5C0B 03
5C0C 04
5C0D 05
5C0E 06
5C0F 07
5C10 08
5C11 09
5C12 00     2200          .BYTE 0,1,2,3,4,5,6,7,8,9
5C13 01
5C14 02
5C15 03
5C16 04
5C17 05
5C18 06
5C19 07
5C1A 08
5C1B 09
5C1C 00     2210          .BYTE 0,1,2,3,4,5,6,7,8,9
5C1D 01
5C1E 02
5C1F 03
5C20 04
5C21 05
5C22 06
5C23 07
5C24 08
5C25 09
5C26 00     2220          .BYTE 0,1,2,3,4,5,6,7,8,9
5C27 01
5C28 02
5C29 03
5C2A 04
5C2B 05
5C2C 06
5C2D 07
5C2E 08
5C2F 09
5C30 00     2230          .BYTE 0,1,2,3,4,5,6,7,8,9
5C31 01
5C32 02
5C33 03
5C34 04
5C35 05
5C36 06
5C37 07
5C38 08
5C39 09
5C3A 00     2240          .BYTE 0,1,2,3,4,5,6,7,8,9
5C3B 01
5C3C 02
5C3D 03
5C3E 04
5C3F 05
5C40 06
5C41 07
5C42 08
5C43 09
5C44 00     2250          .BYTE 0,1,2,3,4,5,6,7,8,9
5C45 01
5C46 02
5C47 03
5C48 04
5C49 05
5C4A 06
```

```
5C4B 07
5C4C 08
5C4D 09
5C4E 00     2260        .BYTE 0,1,2,3,4,5,6,7,8,9
5C4F 01
5C50 02
5C51 03
5C52 04
5C53 05
5C54 06
5C55 07
5C56 08
5C57 09
5C58 00     2270        .BYTE 0,1,2,3,4,5,6,7,8,9
5C59 01
5C5A 02
5C5B 03
5C5C 04
5C5D 05
5C5E 06
5C5F 07
5C60 08
5C61 09
5C62 00     2280        .BYTE 0,1,2,3,4,5,6,7,8,9
5C63 01
5C64 02
5C65 03
5C66 04
5C67 05
5C68 06
5C69 07
5C6A 08
5C6B 09
5C6C 00     2290        .BYTE 0,1,2,3,4,5,6,7,8,9
5C6D 01
5C6E 02
5C6F 03
5C70 04
5C71 05
5C72 06
5C73 07
5C74 08
5C75 09
5C76 00     2300        .BYTE 0,1,2,3,4,5,6,7,8,9
5C77 01
5C78 02
5C79 03
5C7A 04
5C7B 05
5C7C 06
5C7D 07
5C7E 08
5C7F 09
5C80 00     2310        .BYTE 0,1,2,3,4,5,6,7,8,9
5C81 01
5C82 02
5C83 03
5C84 04
5C85 05
5C86 06
5C87 07
5C88 08
5C89 09
5C8A 00     2320        .BYTE 0,1,2,3,4,5,6,7,8,9
5C8B 01
5C8C 02
5C8D 03
5C8E 04
5C8F 05
5C90 06
5C91 07
5C92 08
5C93 09
5C94 00     2330        .BYTE 0,1,2,3,4,5,6,7,8,9
5C95 01
5C96 02
5C97 03
5C98 04
5C99 05
5C9A 06
5C9B 07
5C9C 08
5C9D 09
5C9E 00     2340        .BYTE 0,1,2,3,4,5,6,7,8,9
5C9F 01
5CA0 02
5CA1 03
5CA2 04
5CA3 05
5CA4 06
5CA5 07
5CA6 08
5CA7 09
5CA8 00     2350        .BYTE 0,1,2,3,4,5,6,7,8,9
5CA9 01
5CAA 02
5CAB 03
5CAC 04
5CAD 05
5CAE 06
5CAF 07
5CB0 08
5CB1 09
5CB2 00     2360        .BYTE 0,1,2,3,4,5,6,7,8,9
```

```
5CB3 01
5CB4 02
5CB5 03
5CB6 04
5CB7 05
5CB8 06
5CB9 07
5CBA 08
5CBB 09
5CBC 00     2370        .BYTE 0,1,2,3,4,5,6,7,8,9
5CBD 01
5CBE 02
5CBF 03
5CC0 04
5CC1 05
5CC2 06
5CC3 07
5CC4 08
5CC5 09
5CC6 00     2380        .BYTE 0,1,2,3,4,5,6,7,8,9
5CC7 01
5CC8 02
5CC9 03
5CCA 04
5CCB 05
5CCC 06
5CCD 07
5CCE 08
5CCF 09
5CD0 00     2390        .BYTE 0,1,2,3,4,5,6,7,8,9
5CD1 01
5CD2 02
5CD3 03
5CD4 04
5CD5 05
5CD6 06
5CD7 07
5CD8 08
5CD9 09
5CDA 00     2400        .BYTE 0,1,2,3,4,5,6,7,8,9
5CDB 01
5CDC 02
5CDD 03
5CDE 04
5CDF 05
5CE0 06
5CE1 07
5CE2 08
5CE3 09
5CE4 00     2410        .BYTE 0,1,2,3,4,5,6,7,8,9
5CE5 01
5CE6 02
5CE7 03
5CE8 04
5CE9 05
5CEA 06
5CEB 07
5CEC 08
5CED 09
5CEE 00     2420        .BYTE 0,1,2,3,4,5,6,7,8,9
5CEF 01
5CF0 02
5CF1 03
5CF2 04
5CF3 05
5CF4 06
5CF5 07
5CF6 08
5CF7 09
5CF8 00     2430        .BYTE 0,1,2,3,4,5,6,7,8,9
5CF9 01
5CFA 02
5CFB 03
5CFC 04
5CFD 05
5CFE 06
5CFF 07
5D00 08
5D01 09
5D02 00     2440        .BYTE 0,1,2,3,4,5
5D03 01
5D04 02
5D05 03
5D06 04
5D07 05
5D08 50     2450 TXTTBL .BYTE "PLEASE ENTER YOU"
5D09 4C
5D0A 45
5D0B 41
5D0C 53
5D0D 45
5D0E 20
5D0F 45
5D10 4E
5D11 54
5D12 45
5D13 52
5D14 20
5D15 59
5D16 4F
5D17 55
5D18 52     2460        .BYTE "R ORDERS NOW    "
5D19 20
5D1A 4F
```

```
5D1B 52
5D1C 44
5D1D 45
5D1E 52
5D1F 53
5D20 20
5D21 4E
5D22 4F
5D23 57
5D24 20
5D25 20
5D26 20
5D27 20
5D28 20     2470        .BYTE "          GAME O"
5D29 20
5D2A 20
5D2B 20
5D2C 20
5D2D 20
5D2E 20
5D2F 20
5D30 20
5D31 20
5D32 47
5D33 41
5D34 4D
5D35 45
5D36 20
5D37 4F
5D38 56     2480        .BYTE "VER              "
5D39 45
5D3A 52
5D3B 20
5D3C 20
5D3D 20
5D3E 20
5D3F 20
5D40 20
5D41 20
5D42 20
5D43 20
5D44 20
5D45 20
5D46 20
5D47 20
5D48 46     2490        .BYTE "FIGURING MOVE; N"
5D49 49
5D4A 47
5D4B 55
5D4C 52
5D4D 49
5D4E 4E
5D4F 47
5D50 20
5D51 4D
5D52 4F
5D53 56
5D54 45
5D55 3B
5D56 20
5D57 4E
5D58 4F     2500        .BYTE "O ORDERS ALLOWED"
5D59 20
5D5A 4F
5D5B 52
5D5C 44
5D5D 45
5D5E 52
5D5F 53
5D60 20
5D61 41
5D62 4C
5D63 4C
5D64 4F
5D65 57
5D66 45
5D67 44
5D68 00     2510 MONLEN .BYTE 0,31,28,31,30,31
5D69 1F
5D6A 1C
5D6B 1F
5D6C 1E
5D6D 1F
5D6E 1E     2520        .BYTE 30,31,31,30,31,30,31
5D6F 1F
5D70 1F
5D71 1E
5D72 1F
5D73 1E
5D74 1F
5D75        2530 HMORDS *=   *+159
5E14        2540 WMORDS *=   *+318
5F52 1E     2550 BEEPTB .BYTE 30,40,50,60
5F53 28
5F54 32
5F55 3C
5F56 20     2560 ERRMSG .BYTE "    THAT IS A RU"
5F57 20
5F58 20
5F59 20
5F5A 54
5F5B 48
5F5C 41
5F5D 54
```

```
5F5E 20
5F5F 49
5F60 53
5F61 20
5F62 41
5F63 20
5F64 52
5F65 55
5F66 53     2570        .BYTE "SSIAN UNIT!     "
5F67 53
5F68 49
5F69 41
5F6A 4E
5F6B 20
5F6C 55
5F6D 4E
5F6E 49
5F6F 54
5F70 21
5F71 20
5F72 20
5F73 20
5F74 20
5F75 20
5F76 20     2580        .BYTE "   ONLY 8 ORDERS"
5F77 20
5F78 20
5F79 4F
5F7A 4E
5F7B 4C
5F7C 59
5F7D 20
5F7E 38
5F7F 20
5F80 4F
5F81 52
5F82 44
5F83 45
5F84 52
5F85 53
5F86 20     2590        .BYTE " ARE ALLOWED!   "
5F87 41
5F88 52
5F89 45
5F8A 20
5F8B 41
5F8C 4C
5F8D 4C
5F8E 4F
5F8F 57
5F90 45
5F91 44
```

```
5F92 21
5F93 20
5F94 20
5F95 20
5F96 20     2600        .BYTE "  PLEASE WAIT FO"
5F97 20
5F98 50
5F99 4C
5F9A 45
5F9B 41
5F9C 53
5F9D 45
5F9E 20
5F9F 57
5FA0 41
5FA1 49
5FA2 54
5FA3 20
5FA4 46
5FA5 4F
5FA6 52     2610        .BYTE "R MALTAKREUZE!  "
5FA7 20
5FA8 4D
5FA9 41
5FAA 4C
5FAB 54
5FAC 41
5FAD 4B
5FAE 52
5FAF 45
5FB0 55
5FB1 5A
5FB2 45
5FB3 21
5FB4 20
5FB5 20
5FB6 20     2620        .BYTE "   NO DIAGONAL M"
5FB7 20
5FB8 20
5FB9 4E
5FBA 4F
5FBB 20
5FBC 44
5FBD 49
5FBE 41
5FBF 47
5FC0 4F
5FC1 4E
5FC2 41
5FC3 4C
5FC4 20
5FC5 4D
```

```
5FC6 4F     2630        .BYTE "OVES ALLOWED!   "
5FC7 56
5FC8 45
5FC9 53
5FCA 20
5FCB 41
5FCC 4C
5FCD 4C
5FCE 4F
5FCF 57
5FD0 45
5FD1 44
5FD2 21
5FD3 20
5FD4 20
5FD5 20
5FD6 00     2640 XOFF   .BYTE 0,8,0,$F8
5FD7 08
5FD8 00
5FD9 F8
5FDA F8     2650 YOFF   .BYTE $F8,0,8,0
5FDB 00
5FDC 08
5FDD 00
5FDE 03     2660 MASKO  .BYTE 3,$0C,$30,$C0
5FDF 0C
5FE0 30
5FE1 C0
5FE2 00     2670 XADD   .BYTE 0,1,0,$FF
5FE3 01
5FE4 00
5FE5 FF
5FE6 FF     2680 YADD   .BYTE $FF,0,1,0
5FE7 00
5FE8 01
5FE9 00
5FEA 00     2690 TRTAB  .BYTE 0,$12,$12,$12,$D2,$D8
5FEB 12
5FEC 12
5FED 12
5FEE D2
5FEF D8
5FF0 D6     2700        .BYTE $D6,$C4,$D4,$C2,$12,$12,$12
5FF1 C4
5FF2 D4
5FF3 C2
5FF4 12
5FF5 12
5FF6 12
5FF7 24     2710 MLTKRZ .BYTE $24,$24,$E7,0,0,$E7,$24,$24
5FF8 24
5FF9 E7
5FFA 00
5FFB 00
5FFC E7
5FFD 24
5FFE 24
5FFF        2720        *=   $6000
            2730 ;First comes 1024 bytes of new character set
6000        2740        *=   *+1024
            2750 ;
            2760 ;The display list goes here; it is 49 bytes long.
            2770 ;
6400 70     2780        .BYTE $70,$70,$70,$C6,$E0,$64,$90,$90,$F7
6401 70
6402 70
6403 C6
6404 E0
6405 64
6406 90
6407 90
6408 F7
6409 FE d790        .BYTE $FE,$64,$F7,$2E,$65,$F7,$5E,$65
640A 64
640B F7
640C 2E
640D 65
640E F7
640F 5E
6410 65
6411 F7     2800        .BYTE $F7,$8E,$65,$F7,$BE,$65,$F7,$EE
6412 8E
6413 65
6414 F7
6415 BE
6416 65
6417 F7
6418 EE
6419 65     2810        .BYTE $65,$F7,$1E,$66,$F7,$4E,$66,$F7
641A F7
641B 1E
641C 66
641D F7
641E 4E
641F 66
6420 F7
6421 7E     2820        .BYTE $7E,$66,$57,$AE,$66,$90,$C2,$50
6422 66
6423 57
6424 AE
6425 66
6426 90
6427 C2
6428 50
```

```
6429 64     2830         .BYTE $64,$02,$90,$02,$90,$41,$00,$64
642A 02
642B 90
642C 02
642D 90
642E 41
642F 00
6430 64
6431 10     2840 ARRTAB .BYTE $10,$38,$54,$92,$10,$10,$10,$10
6432 38
6433 54
6434 92
6435 10
6436 10
6437 10
6438 10
6439 08     2850         .BYTE 8,4,2,$FF,2,4,8,0
643A 04
643B 02
643C FF
643D 02
643E 04
643F 08
6440 00
6441 10     2860         .BYTE $10,$10,$10,$10,$92,$54,$38,$10
6442 10
6443 10
6444 10
6445 92
6446 54
6447 38
6448 10
6449 10     2870         .BYTE $10,$20,$40,$FF,$40,$20,$10,0
644A 20
644B 40
644C FF
644D 40
644E 20
644F 10
6450 00
            2880 ;
6451        2890         *=   $6450
            2900 ;This next area is reserved for the text window
6450        2910 TXTWDW *=   $64FF
            2920 ;
            2930 ;The map data goes here.
            2940 ;
64FF 7F     2950         .BYTE 127,127,127,127,127,127,127,127,127
6500 7F
6501 7F
6502 7F
6503 7F
6504 7F
6505 7F
6506 7F
6507 7F
6508 7F     2960         .BYTE 127,127,127,127,127,127,127,127
6509 7F
650A 7F
650B 7F
650C 7F
650D 7F
650E 7F
650F 7F
6510 7F     2970         .BYTE 127,127,127,127,127,127,127,127
6511 7F
6512 7F
6513 7F
6514 7F
6515 7F
6516 7F
6517 7F
6518 7F     2980         .BYTE 127,127,127,127,127,127,127,127
6519 7F
651A 7F
651B 7F
651C 7F
651D 7F
651E 7F
651F 7F
6520 7F     2990         .BYTE 127,127,127,127,127,127,127,127
6521 7F
6522 7F
6523 7F
6524 7F
6525 7F
6526 7F
6527 7F
6528 7F     3000         .BYTE 127,127,127,127,127,127,127,127
6529 7F
652A 7F
652B 7F
652C 7F
652D 7F
652E 7F
652F 7F
6530 7F     3010         .BYTE 127,191,191,191,169,0,0,0
6531 BF
6532 BF
6533 BF
6534 A9
6535 00
6536 00
6537 00
```

```
6538 00     3020        .BYTE 0,0,0,0,0,180,191,191
6539 00
653A 00
653B 00
653C 00
653D B4
653E BF
653F BF
6540 AA     3030        .BYTE 170,0,0,0,0,0,0,0
6541 00
6542 00
6543 00
6544 00
6545 00
6546 00
6547 00
6548 00     3040        .BYTE 0,0,0,0,0,0,0,0
6549 00
654A 00
654B 00
654C 00
654D 00
654E 00
654F 00
6550 00     3050        .BYTE 0,0,0,0,0,0,0,0
6551 00
6552 00
6553 00
6554 00
6555 00
6556 00
6557 00
6558 00     3060        .BYTE 0,0,0,0,0,0,0,127
6559 00
655A 00
655B 00
655C 00
655D 00
655E 00
655F 7F
6560 7F     3070        .BYTE 127,191,191,191,175,178,0,0
6561 BF
6562 BF
6563 BF
6564 AF
6565 B2
6566 00
6567 00
6568 00     3080        .BYTE 0,181,182,184,183,182,179,187
6569 B5
656A B6
656B B8
656C B7
656D B6
656E B3
656F BB
6570 B0     3090        .BYTE 176,0,0,0,0,0,0,0
6571 00
6572 00
6573 00
6574 00
6575 00
6576 00
6577 00
6578 00     3100        .BYTE 0,0,0,0,0,0,0,0
6579 00
657A 00
657B 00
657C 00
657D 00
657E 00
657F 00
6580 00     3110        .BYTE 0,0,0,0,0,0,0,0
6581 00
6582 00
6583 00
6584 00
6585 00
6586 00
6587 00
6588 00     3120        .BYTE 0,0,0,0,0,0,0,127
6589 00
658A 00
658B 00
658C 00
658D 00
658E 00
658F 7F
6590 7F     3130        .BYTE 127,191,191,191,191,175,184,183
6591 BF
6592 BF
6593 BF
6594 BF
6595 AF
6596 B8
6597 B7
6598 B9     3140        .BYTE 185,191,191,177,176,71,157,155
6599 BF
659A BF
659B B1
659C B0
659D 47
659E 9D
659F 9B
```

```
65A0 00     3150        .BYTE 0,0,0,0,0,0,0,0
65A1 00
65A2 00
65A3 00
65A4 00
65A5 00
65A6 00
65A7 00
65A8 00     3160        .BYTE 0,0,0,0,0,0,0,0
65A9 00
65AA 00
65AB 00
65AC 00
65AD 00
65AE 00
65AF 00
65B0 00     3170        .BYTE 0,0,0,0,0,0,0,0
65B1 00
65B2 00
65B3 00
65B4 00
65B5 00
65B6 00
65B7 00
65B8 00     3180        .BYTE 0,0,0,0,0,0,0,127
65B9 00
65BA 00
65BB 00
65BC 00
65BD 00
65BE 00
65BF 7F
65C0 7F     3190        .BYTE 127,191,191,191,191,191,177,172
65C1 BF
65C2 BF
65C3 BF
65C4 BF
65C5 BF
65C6 B1
65C7 AC
65C8 AD     3200        .BYTE 173,174,187,188,164,141,148,140
65C9 AE
65CA BB
65CB BC
65CC A4
65CD 8D
65CE 94
65CF 8C
65D0 00     3210        .BYTE 0,0,0,0,0,0,0,0
65D1 00
65D2 00
65D3 00
65D4 00
65D5 00
65D6 00
65D7 00
65D8 9D     3220        .BYTE 157,165,0,156,160,162,166,0
65D9 A5
65DA 00
65DB 9C
65DC A0
65DD A2
65DE A6
65DF 00
65E0 00     3230        .BYTE 0,0,0,0,0,0,0,0
65E1 00
65E2 00
65E3 00
65E4 00
65E5 00
65E6 00
65E7 00
65E8 00     3240        .BYTE 0,0,0,0,0,0,0,127
65E9 00
65EA 00
65EB 00
65EC 00
65ED 00
65EE 00
65EF 7F
65F0 7F     3250        .BYTE 127,191,191,191,191,191,171,0
65F1 BF
65F2 BF
65F3 BF
65F4 BF
65F5 BF
65F6 AB
65F7 00
65F8 00     3260        .BYTE 0,0,186,178,152,142,149,1
65F9 00
65FA BA
65FB B2
65FC 98
65FD 8E
65FE 95
65FF 01
6600 05     3270        .BYTE 5,0,0,0,0,0,0,0
6601 00
6602 00
6603 00
6604 00
6605 00
6606 00
6607 00
```

```
6608 94    3280        .BYTE 148,145,161,154,0,0,146,159
6609 91
660A A1
660B 9A
660C 00
660D 00
660E 92
660F 9F
6610 A5    3290        .BYTE 165,0,0,0,0,156,164,0
6611 00
6612 00
6613 00
6614 00
6615 9C
6616 A4
6617 00
6618 00    3300        .BYTE 0,0,0,0,0,0,0,127
6619 00
661A 00
661B 00
661C 00
661D 00
661E 00
661F 7F
6620 7F    3310        .BYTE 127,191,191,191,191,191,170,0
6621 BF
6622 BF
6623 BF
6624 BF
6625 BF
6626 AA
6627 00
6628 00    3320        .BYTE 0,0,180,170,147,140,150,2
6629 00
662A B4
662B AA
662C 93
662D 8C
662E 96
662F 02
6630 06    3330        .BYTE 6,0,0,0,0,0,0,0
6631 00
6632 00
6633 00
6634 00
6635 00
6636 00
6637 00
6638 97    3340        .BYTE 151,0,0,0,0,0,0,156
6639 00
663A 00
663B 00
663C 00
663D 00
663E 00
663F 9C
6640 A8    3350        .BYTE 168,72,0,157,161,153,145,160
6641 48
6642 00
6643 9D
6644 A1
6645 99
6646 91
6647 A0
6648 A5    3360        .BYTE 165,0,0,0,0,0,0,127
6649 00
664A 00
664B 00
664C 00
664D 00
664E 00
664F 7F
6650 7F    3370        .BYTE 127,191,191,191,191,191,175,178
6651 BF
6652 BF
6653 BF
6654 BF
6655 BF
6656 AF
6657 B2
6658 00    3380        .BYTE 0,0,0,176,149,139,151,3
6659 00
665A 00
665B B0
665C 95
665D 8B
665E 97
665F 03
6660 01    3390        .BYTE 1,0,0,0,0,0,0,0
6661 00
6662 00
6663 00
6664 00
6665 00
6666 00
6667 00
6668 00    3400        .BYTE 0,0,0,0,0,0,0,149
6669 00
666A 00
666B 00
666C 00
666D 00
666E 00
666F 95
```

```
6670 91     3410        .BYTE 145,160,159,155,0,0,0,73
6671 A0
6672 9F
6673 9B
6674 00
6675 00
6676 00
6677 49
6678 92     3420        .BYTE 146,166,0,0,0,0,0,127
6679 A6
667A 00
667B 00
667C 00
667D 00
667E 00
667F 7F
6680 7F     3430        .BYTE 127,191,191,191,191,191,191,169
6681 BF
6682 BF
6683 BF
6684 BF
6685 BF
6686 BF
6687 A9
6688 00     3440        .BYTE 0,0,0,0,0,0,152,4
6689 00
668A 00
668B 00
668C 00
668D 00
668E 98
668F 04
6690 03     3450        .BYTE 3,0,0,0,0,0,0,0
6691 00
6692 00
6693 00
6694 00
6695 00
6696 00
6697 00
6698 00     3460        .BYTE 0,0,0,0,0,0,157,154
6699 00
669A 00
669B 00
669C 00
669D 00
669E 9D
669F 9A
66A0 00     3470        .BYTE 0,0,0,0,0,0,0,0
66A1 00
66A2 00
66A3 00
66A4 00
66A5 00
66A6 00
66A7 00
66A8 00     3480        .BYTE 0,149,0,0,0,0,0,127
66A9 95
66AA 00
66AB 00
66AC 00
66AD 00
66AE 00
66AF 7F
66B0 7F     3490        .BYTE 127,191,191,177,172,191,191,170
66B1 BF
66B2 BF
66B3 B1
66B4 AC
66B5 BF
66B6 BF
66B7 AA
66B8 48     3500        .BYTE 72,0,0,0,0,0,148,0
66B9 00
66BA 00
66BB 00
66BC 00
66BD 00
66BE 94
66BF 00
66C0 02     3510        .BYTE 2,0,0,0,0,0,0,0
66C1 00
66C2 00
66C3 00
66C4 00
66C5 00
66C6 00
66C7 00
66C8 02     3520        .BYTE 2,0,0,0,0,0,150,0
66C9 00
66CA 00
66CB 00
66CC 00
66CD 00
66CE 96
66CF 00
66D0 00     3530        .BYTE 0,0,0,0,0,0,0,0
66D1 00
66D2 00
66D3 00
66D4 00
66D5 00
66D6 00
66D7 00
```

```
66D8 00     3540        .BYTE 0,144,162,159,167,0,0,127
66D9 90
66DA A2
66DB 9F
66DC A7
66DD 00
66DE 00
66DF 7F
66E0 7F     3550        .BYTE 127,191,191,170,0,179,173,188
66E1 BF
66E2 BF
66E3 AA
66E4 00
66E5 B3
66E6 AD
66E7 BC
66E8 9F     3560        .BYTE 159,160,165,0,0,0,0,0
66E9 A0
66EA A5
66EB 00
66EC 00
66ED 00
66EE 00
66EF 00
66F0 00     3570        .BYTE 0,0,0,0,0,0,0,0
66F1 00
66F2 00
66F3 00
66F4 00
66F5 00
66F6 00
66F7 00
66F8 01     3580        .BYTE 1,0,0,0,0,0,151,0
66F9 00
66FA 00
66FB 00
66FC 00
66FD 00
66FE 97
66FF 00
6700 00     3590        .BYTE 0,0,0,0,0,0,0,0
6701 00
6702 00
6703 00
6704 00
6705 00
6706 00
6707 00
6708 00     3600        .BYTE 0,0,0,156,153,0,0,127
6709 00
670A 00
670B 9C
670C 99
670D 00
670E 00
670F 7F
6710 7F     3610        .BYTE 127,191,191,169,0,0,0,0
6711 BF
6712 BF
6713 A9
6714 00
6715 00
6716 00
6717 00
6718 00     3620        .BYTE 0,0,143,164,0,0,0,0
6719 00
671A 8F
671B A4
671C 00
671D 00
671E 00
671F 00
6720 00     3630        .BYTE 0,157,155,0,0,0,73,0
6721 9D
6722 9B
6723 00
6724 00
6725 00
6726 49
6727 00
6728 00     3640        .BYTE 0,0,74,0,0,156,153,0
6729 00
672A 4A
672B 00
672C 00
672D 9C
672E 99
672F 00
6730 00     3650        .BYTE 0,0,0,0,0,0,0,0
6731 00
6732 00
6733 00
6734 00
6735 00
6736 00
6737 00
6738 00     3660        .BYTE 0,0,0,149,0,0,0,127
6739 00
673A 00
673B 95
673C 00
673D 00
673E 00
673F 7F
```

```
6740 7F     3670        .BYTE 127,191,191,171,0,0,0,0
6741 BF
6742 BF
6743 AB
6744 00
6745 00
6746 00
6747 00
6748 00     3680        .BYTE 0,0,0,144,161,166,0,0
6749 00
674A 00
674B 90
674C A1
674D A6
674E 00
674F 00
6750 9C     3690        .BYTE 156,154,0,0,0,0,0,3
6751 9A
6752 00
6753 00
6754 00
6755 00
6756 00
6757 03
6758 06     3700        .BYTE 6,0,0,0,0,152,0,0
6759 00
675A 00
675B 00
675C 00
675D 98
675E 00
675F 00
6760 00     3710        .BYTE 0,0,0,0,0,0,0,0
6761 00
6762 00
6763 00
6764 00
6765 00
6766 00
6767 00
6768 00     3720        .BYTE 0,0,0,147,0,0,0,127
6769 00
676A 00
676B 93
676C 00
676D 00
676E 00
676F 7F
6770 7F     3730        .BYTE 127,191,191,175,178,0,0,0
6771 BF
6772 BF
6773 AF
6774 B2
6775 00
6776 00
6777 00
6778 00     3740        .BYTE 0,0,0,0,0,145,162,163
6779 00
677A 00
677B 00
677C 00
677D 91
677E A2
677F A3
6780 99     3750        .BYTE 153,0,0,0,2,151,4,1
6781 00
6782 00
6783 00
6784 02
6785 97
6786 04
6787 01
6788 02     3760        .BYTE 2,158,163,161,159,155,0,0
6789 9E
678A A3
678B A1
678C 9F
678D 9B
678E 00
678F 00
6790 00     3770        .BYTE 0,0,0,0,0,0,0,0
6791 00
6792 00
6793 00
6794 00
6795 00
6796 00
6797 00
6798 00     3780        .BYTE 0,0,0,150,0,0,0,127
6799 00
679A 00
679B 96
679C 00
679D 00
679E 00
679F 7F
67A0 7F     3790        .BYTE 127,191,191,191,170,0,0,0
67A1 BF
67A2 BF
67A3 BF
67A4 AA
67A5 00
67A6 00
67A7 00
```

```
67A8 00     3800        .BYTE 0,0,0,0,0,0,0,0
67A9 00
67AA 00
67AB 00
67AC 00
67AD 00
67AE 00
67AF 00
67B0 00     3810        .BYTE 0,0,0,156,162,153,0,3
67B1 00
67B2 00
67B3 9C
67B4 A2
67B5 99
67B6 00
67B7 03
67B8 04     3820        .BYTE 4,148,0,0,0,0,0,0
67B9 94
67BA 00
67BB 00
67BC 00
67BD 00
67BE 00
67BF 00
67C0 00     3830        .BYTE 0,0,0,0,0,0,0,0
67C1 00
67C2 00
67C3 00
67C4 00
67C5 00
67C6 00
67C7 00
67C8 00     3840        .BYTE 0,0,156,154,0,0,0,127
67C9 00
67CA 9C
67CB 9A
67CC 00
67CD 00
67CE 00
67CF 7F
67D0 7F     3850        .BYTE 127,191,191,177,188,160,159,161
67D1 BF
67D2 BF
67D3 B1
67D4 BC
67D5 A0
67D6 9F
67D7 A1
67D8 A4     3860        .BYTE 164,0,0,0,2,6,5,0
67D9 00
67DA 00
67DB 00
67DC 02
67DD 06
67DE 05
67DF 00
67E0 00     3870        .BYTE 0,157,163,154,71,0,1,6
67E1 9D
67E2 A3
67E3 9A
67E4 47
67E5 00
67E6 01
67E7 06
67E8 00     3880        .BYTE 0,147,0,0,152,0,0,0
67E9 93
67EA 00
67EB 00
67EC 98
67ED 00
67EE 00
67EF 00
67F0 00     3890        .BYTE 0,0,0,0,0,0,0,0
67F1 00
67F2 00
67F3 00
67F4 00
67F5 00
67F6 00
67F7 00
67F8 00     3900        .BYTE 0,0,151,74,0,0,0,127
67F9 00
67FA 97
67FB 4A
67FC 00
67FD 00
67FE 00
67FF 7F
6800 7F     3910        .BYTE 127,191,177,176,0,0,0,0
6801 BF
6802 B1
6803 B0
6804 00
6805 00
6806 00
6807 00
6808 91     3920        .BYTE 145,162,0,1,4,3,1,0
6809 A2
680A 00
680B 01
680C 04
680D 03
680E 01
680F 00
```

```
6810 9E     3930        .BYTE 158,155,0,0,0,0,0,0
6811 9B
6812 00
6813 00
6814 00
6815 00
6816 00
6817 00
6818 00     3940        .BYTE 0,0,0,0,151,0,0,0
6819 00
681A 00
681B 00
681C 97
681D 00
681E 00
681F 00
6820 00     3950        .BYTE 0,0,0,0,0,0,0,0
6821 00
6822 00
6823 00
6824 00
6825 00
6826 00
6827 00
6828 00     3960        .BYTE 0,0,148,0,0,0,0,127
6829 00
682A 94
682B 00
682C 00
682D 00
682E 00
682F 7F
6830 7F     3970        .BYTE 127,173,176,0,0,0,0,0
6831 AD
6832 B0
6833 00
6834 00
6835 00
6836 00
6837 00
6838 00     3980        .BYTE 0,0,0,2,6,74,0,140
6839 00
683A 00
683B 02
683C 06
683D 4A
683E 00
683F 8C
6840 96     3990        .BYTE 150,139,0,0,0,0,0,0
6841 8B
6842 00
6843 00
6844 00
6845 00
6846 00
6847 00
6848 00     4000        .BYTE 0,0,0,0,143,162,167,0
6849 00
684A 00
684B 00
684C 8F
684D A2
684E A7
684F 00
6850 00     4010        .BYTE 0,0,0,0,0,0,0,0
6851 00
6852 00
6853 00
6854 00
6855 00
6856 00
6857 00
6858 00     4020        .BYTE 0,158,155,0,0,0,0,127
6859 9E
685A 9B
685B 00
685C 00
685D 00
685E 00
685F 7F
6860 7F     4030        .BYTE 127,0,0,0,0,0,0,0
6861 00
6862 00
6863 00
6864 00
6865 00
6866 00
6867 00
6868 00     4040        .BYTE 0,1,3,5,0,0,0,142
6869 01
686A 03
686B 05
686C 00
686D 00
686E 00
686F 8E
6870 90     4050        .BYTE 144,165,141,0,0,0,0,0
6871 A5
6872 8D
6873 00
6874 00
6875 00
6876 00
6877 00
```

```
6878 00     4060        .BYTE 0,71,0,0,0,0,150,73
6879 47
687A 00
687B 00
687C 00
687D 00
687E 96
687F 49
6880 00     4070        .BYTE 0,0,0,0,0,0,0,0
6881 00
6882 00
6883 00
6884 00
6885 00
6886 00
6887 00
6888 00     4080        .BYTE 0,152,0,0,0,0,0,127
6889 98
688A 00
688B 00
688C 00
688D 00
688E 00
688F 7F
6890 7F     4090        .BYTE 127,0,0,0,0,0,0,0
6891 00
6892 00
6893 00
6894 00
6895 00
6896 00
6897 00
6898 02     4100        .BYTE 2,6,0,0,0,0,141,139
6899 06
689A 00
689B 00
689C 00
689D 00
689E 8D
689F 8B
68A0 8E     4110        .BYTE 142,146,167,0,0,0,0,0
68A1 92
68A2 A7
68A3 00
68A4 00
68A5 00
68A6 00
68A7 00
68A8 00     4120        .BYTE 0,0,0,0,0,0,145,165
68A9 00
68AA 00
68AB 00
68AC 00
68AD 00
68AE 91
68AF A5
68B0 00     4130        .BYTE 0,0,0,0,0,0,0,0
68B1 00
68B2 00
68B3 00
68B4 00
68B5 00
68B6 00
68B7 00
68B8 00     4140        .BYTE 0,150,0,0,0,0,0,127
68B9 96
68BA 00
68BB 00
68BC 00
68BD 00
68BE 00
68BF 7F
68C0 7F     4150        .BYTE 127,166,73,0,0,0,0,0
68C1 A6
68C2 49
68C3 00
68C4 00
68C5 00
68C6 00
68C7 00
68C8 05     4160        .BYTE 5,4,0,0,139,140,142,141
68C9 04
68CA 00
68CB 00
68CC 8B
68CD 8C
68CE 8E
68CF 8D
68D0 8C     4170        .BYTE 140,0,152,0,0,0,0,0
68D1 00
68D2 98
68D3 00
68D4 00
68D5 00
68D6 00
68D7 00
68D8 00     4180        .BYTE 0,0,0,0,0,0,0,149
68D9 00
68DA 00
68DB 00
68DC 00
68DD 00
68DE 00
68DF 95
```

```
68E0 00     4190        .BYTE 0,0,0,0,0,0,0,0
68E1 00
68E2 00
68E3 00
68E4 00
68E5 00
68E6 00
68E7 00
68E8 00     4200        .BYTE 0,149,0,0,0,0,0,127
68E9 95
68EA 00
68EB 00
68EC 00
68ED 00
68EE 00
68EF 7F
68F0 7F     4210        .BYTE 127,146,165,0,0,0,0,0
68F1 92
68F2 A5
68F3 00
68F4 00
68F5 00
68F6 00
68F7 00
68F8 03     4220        .BYTE 3,1,0,0,141,159,163,165
68F9 01
68FA 00
68FB 00
68FC 8D
68FD 9F
68FE A3
68FF A5
6900 8E     4230        .BYTE 142,139,148,0,0,0,0,0
6901 8B
6902 94
6903 00
6904 00
6905 00
6906 00
6907 00
6908 00     4240        .BYTE 0,0,150,0,0,0,0,144
6909 00
690A 96
690B 00
690C 00
690D 00
690E 00
690F 90
6910 A1     4250        .BYTE 161,164,0,0,0,0,0,0
6911 A4
6912 00
6913 00
6914 00
6915 00
6916 00
6917 00
6918 00     4260        .BYTE 0,151,0,0,0,0,0,127
6919 97
691A 00
691B 00
691C 00
691D 00
691E 00
691F 7F
6920 7F     4270        .BYTE 127,0,143,167,0,0,0,3
6921 00
6922 8F
6923 A7
6924 00
6925 00
6926 00
6927 03
6928 04     4280        .BYTE 4,6,0,139,140,142,141,145
6929 06
692A 00
692B 8B
692C 8C
692D 8E
692E 8D
692F 91
6930 A0     4290        .BYTE 160,166,151,0,0,0,0,0
6931 A6
6932 97
6933 00
6934 00
6935 00
6936 a37 00
6938 00     4300        .BYTE 0,0,145,166,0,0,0,0
6939 00
693A 91
693B A6
693C 00
693D 00
693E 00
693F 00
6940 00     4310        .BYTE 0,146,166,0,0,0,0,0
6941 92
6942 A6
6943 00
6944 00
6945 00
6946 00
6947 00
```

```
6948 00     4320        .BYTE 0,148,0,0,0,0,0,127
6949 94
694A 00
694B 00
694C 00
694D 00
694E 00
694F 7F
6950 7F     4330        .BYTE 127,0,0,149,0,0,0,2
6951 00
6952 00
6953 95
6954 00
6955 00
6956 00
6957 02
6958 05     4340        .BYTE 5,139,142,141,139,140,139,142
6959 8B
695A 8E
695B 8D
695C 8B
695D 8C
695E 8B
695F 8E
6960 8C     4350        .BYTE 140,146,168,0,0,0,0,0
6961 92
6962 A8
6963 00
6964 00
6965 00
6966 00
6967 00
6968 00     4360        .BYTE 0,0,0,151,0,0,0,0
6969 00
696A 00
696B 97
696C 00
696D 00
696E 00
696F 00
6970 00     4370        .BYTE 0,0,143,163,159,161,160,166
6971 00
6972 8F
6973 A3
6974 9F
6975 A1
6976 A0
6977 A6
6978 00     4380        .BYTE 0,152,0,0,0,0,0,127
6979 98
697A 00
697B 00
697C 00
697D 00
697E 00
697F 7F
6980 7F     4390        .BYTE 127,0,156,154,0,0,0,0
6981 00
6982 9C
6983 9A
6984 00
6985 00
6986 00
6987 00
6988 00     4400        .BYTE 0,140,139,141,142,140,0,0
6989 8C
698A 8B
698B 8D
698C 8E
698D 8C
698E 00
698F 00
6990 00     4410        .BYTE 0,139,148,0,0,0,0,0
6991 8B
6992 94
6993 00
6994 00
6995 00
6996 00
6997 00
6998 00     4420        .BYTE 0,0,74,148,0,0,0,0
6999 00
699A 4A
699B 94
699C 00
699D 00
699E 00
699F 00
69A0 00     4430        .BYTE 0,0,0,0,0,0,0,147
69A1 00
69A2 00
69A3 00
69A4 00
69A5 00
69A6 00
69A7 93
69A8 47     4440        .BYTE 71,143,159,160,162,165,0,127
69A9 8F
69AA 9F
69AB A0
69AC A2
69AD A5
69AE 00
69AF 7F
```

```
69B0 7F     4450        .BYTE 127,153,151,0,0,0,0,0
69B1 99
69B2 97
69B3 00
69B4 00
69B5 00
69B6 00
69B7 00
69B8 00     4460        .BYTE 0,0,142,0,0,0,0,0
69B9 00
69BA 8E
69BB 00
69BC 00
69BD 00
69BE 00
69BF 00
69C0 00     4470        .BYTE 0,71,149,0,0,0,0,0
69C1 47
69C2 95
69C3 00
69C4 00
69C5 00
69C6 00
69C7 00
69C8 00     4480        .BYTE 0,0,0,144,165,0,0,0
69C9 00
69CA 00
69CB 90
69CC A5
69CD 00
69CE 00
69CF 00
69D0 00     4490        .BYTE 0,0,0,0,0,0,0,149
69D1 00
69D2 00
69D3 00
69D4 00
69D5 00
69D6 00
69D7 95
69D8 00     4500        .BYTE 0,0,0,0,0,144,166,127
69D9 00
69DA 00
69DB 00
69DC 00
69DD 90
69DE A6
69DF 7F
69E0 7F     4510        .BYTE 127,1,6,0,0,0,0,0
69E1 01
69E2 06
69E3 00
69E4 00
69E5 00
69E6 00
69E7 00
69E8 00     4520        .BYTE 0,0,0,0,0,0,0,0
69E9 00
69EA 00
69EB 00
69EC 00
69ED 00
69EE 00
69EF 00
69F0 00     4530        .BYTE 0,0,143,156,161,0,0,0
69F1 00
69F2 8F
69F3 9C
69F4 A1
69F5 00
69F6 00
69F7 00
69F8 00     4540        .BYTE 0,0,0,0,146,156,155,157
69F9 00
69FA 00
69FB 00
69FC 92
69FD 9C
69FE 9B
69FF 9D
6A00 9A     4550        .BYTE 154,156,160,0,0,0,0,148
6A01 9C
6A02 A0
6A03 00
6A04 00
6A05 00
6A06 00
6A07 94
6A08 00     4560        .BYTE 0,0,0,0,0,0,146,127
6A09 00
6A0A 00
6A0B 00
6A0C 00
6A0D 00
6A0E 92
6A0F 7F
6A10 7F     4570        .BYTE 127,2,5,3,4,0,0,0
6A11 02
6A12 05
6A13 03
6A14 04
6A15 00
6A16 00
6A17 00
```

```
6A18 00     4560        .BYTE 0,0,0,0,0,0,0,0
6A19 00
6A1A 00
6A1B 00
6A1C 00
6A1D 00
6A1E 00
6A1F 00
6A20 00     4590        .BYTE 0,0,0,0,145,155,158,0
6A21 00
6A22 00
6A23 00
6A24 91
6A25 9B
6A26 9E
6A27 00
6A28 00     4600        .BYTE 0,0,0,0,0,0,0,0
6A29 00
6A2A 00
6A2B 00
6A2C 00
6A2D 00
6A2E 00
6A2F 00
6A30 00     4610        .BYTE 0,0,145,157,158,0,152,150
6A31 00
6A32 91
6A33 9D
6A34 9E
6A35 00
6A36 98
6A37 96
6A38 00     4620        .BYTE 0,0,0,0,0,0,0,127
6A39 00
6A3A 00
6A3B 00
6A3C 00
6A3D 00
6A3E 00
6A3F 7F
6A40 7F     4630        .BYTE 127,0,0,1,5,6,3,0
6A41 00
6A42 00
6A43 01
6A44 05
6A45 06
6A46 03
6A47 00
6A48 00     4640        .BYTE 0,156,161,0,0,0,156,159
6A49 9C
6A4A A1
6A4B 00
6A4C 00
6A4D 00
6A4E 9C
6A4F 9F
6A50 00     4650        .BYTE 0,0,0,0,0,0,144,154
6A51 00
6A52 00
6A53 00
6A54 00
6A55 00
6A56 90
6A57 9A
6A58 A0     4660        .BYTE 160,0,0,0,0,0,0,0
6A59 00
6A5A 00
6A5B 00
6A5C 00
6A5D 00
6A5E 00
6A5F 00
6A60 00     4670        .BYTE 0,0,0,153,162,155,151,0
6A61 00
6A62 00
6A63 99
6A64 A2
6A65 9B
6A66 97
6A67 00
6A68 00     4680        .BYTE 0,0,0,0,0,0,0,127
6A69 00
6A6A 00
6A6B 00
6A6C 00
6A6D 00
6A6E 00
6A6F 7F
6A70 7F     4690        .BYTE 127,0,0,0,0,4,3,1
6A71 00
6A72 00
6A73 00
6A74 00
6A75 04
6A76 03
6A77 01
6A78 05     4700        .BYTE 5,0,145,159,0,0,0,146
6A79 00
6A7A 91
6A7B 9F
6A7C 00
6A7D 00
6A7E 00
6A7F 92
```

```
6A80 9D     4710        .BYTE 157,158,0,0,0,0,0,0
6A81 9E
6A82 00
6A83 00
6A84 00
6A85 00
6A86 00
6A87 00
6A88 92     4720        .BYTE 146,157,159,0,0,0,0,0
6A89 9D
6A8A 9F
6A8B 00
6A8C 00
6A8D 00
6A8E 00
6A8F 00
6A90 00     4730        .BYTE 0,0,152,151,0,0,0,0
6A91 00
6A92 98
6A93 97
6A94 00
6A95 00
6A96 00
6A97 00
6A98 00     4740        .BYTE 0,0,0,0,0,0,0,127
6A99 00
6A9A 00
6A9B 00
6A9C 00
6A9D 00
6A9E 00
6A9F 7F
6AA0 7F     4750        .BYTE 127,0,0,0,0,0,2,4
6AA1 00
6AA2 00
6AA3 00
6AA4 00
6AA5 00
6AA6 02
6AA7 04
6AA8 06     4760        .BYTE 6,0,0,143,155,156,154,160
6AA9 00
6AAA 00
6AAB 8F
6AAC 9B
6AAD 9C
6AAE 9A
6AAF A0
6AB0 00     4770        .BYTE 0,143,154,161,0,0,0,0
6AB1 8F
6AB2 9A
6AB3 A1
6AB4 00
6AB5 00
6AB6 00
6AB7 00
6AB8 00     4780        .BYTE 0,0,143,158,0,0,0,0
6AB9 00
6ABA 8F
6ABB 9E
6ABC 00
6ABD 00
6ABE 00
6ABF 00
6AC0 00     4790        .BYTE 0,153,150,0,0,0,0,0
6AC1 99
6AC2 96
6AC3 00
6AC4 00
6AC5 00
6AC6 00
6AC7 00
6AC8 00     4800        .BYTE 0,0,0,0,0,0,0,127
6AC9 00
6ACA 00
6ACB 00
6ACC 00
6ACD 00
6ACE 00
6ACF 7F
6AD0 7F     4810        .BYTE 127,0,0,0,0,0,0,1
6AD1 00
6AD2 00
6AD3 00
6AD4 00
6AD5 00
6AD6 00
6AD7 01
6AD8 03     4820        .BYTE 3,5,0,0,0,0,0,144
6AD9 05
6ADA 00
6ADB 00
6ADC 00
6ADD 00
6ADE 00
6ADF 90
6AE0 9E     4830        .BYTE 158,0,0,145,160,0,0,0
6AE1 00
6AE2 00
6AE3 91
6AE4 A0
6AE5 00
6AE6 00
6AE7 00
```

```
6AE8 00     4840        .BYTE 0,0,72,147,0,0,0,176
6AE9 00
6AEA 48
6AEB 93
6AEC 00
6AED 00
6AEE 00
6AEF B0
6AF0 A5     4850        .BYTE 165,188,73,0,0,0,0,0
6AF1 BC
6AF2 49
6AF3 00
6AF4 00
6AF5 00
6AF6 00
6AF7 00
6AF8 00     4860        .BYTE 0,0,0,0,0,0,0,127
6AF9 00
6AFA 00
6AFB 00
6AFC 00
6AFD 00
6AFE 00
6AFF 7F
6B00 7F     4870        .BYTE 127,0,0,0,0,0,0,0
6B01 00
6B02 00
6B03 00
6B04 00
6B05 00
6B06 00
6B07 00
6B08 02     4880        .BYTE 2,6,4,0,0,0,0,0
6B09 06
6B0A 04
6B0B 00
6B0C 00
6B0D 00
6B0E 00
6B0F 00
6B10 92     4890        .BYTE 146,161,0,0,144,159,0,0
6B11 A1
6B12 00
6B13 00
6B14 90
6B15 9F
6B16 00
6B17 00
6B18 00     4900        .BYTE 0,0,153,150,0,177,166,170
6B19 00
6B1A 99
6B1B 96
6B1C 00
6B1D B1
6B1E A6
6B1F AA
6B20 B2     4910        .BYTE 178,174,0,0,0,0,0,0
6B21 AE
6B22 00
6B23 00
6B24 00
6B25 00
6B26 00
6B27 00
6B28 00     4920        .BYTE 0,0,0,0,0,0,0,127
6B29 00
6B2A 00
6B2B 00
6B2C 00
6B2D 00
6B2E 00
6B2F 7F
6B30 7F     4930        .BYTE 127,0,0,0,0,0,0,0
6B31 00
6B32 00
6B33 00
6B34 00
6B35 00
6B36 00
6B37 00
6B38 00     4940        .BYTE 0,5,1,6,0,160,0,0
6B39 05
6B3A 01
6B3B 06
6B3C 00
6B3D A0
6B3E 00
6B3F 00
6B40 00     4950        .BYTE 0,143,159,0,0,146,158,0
6B41 8F
6B42 9F
6B43 00
6B44 00
6B45 92
6B46 9E
6B47 00
6B48 00     4960        .BYTE 0,0,149,0,175,171,191,179
6B49 00
6B4A 95
6B4B 00
6B4C AF
6B4D AB
6B4E BF
6B4F B3
```

```
6B50 AD     4970        .BYTE 173,0,0,0,0,0,0,0
6B51 00
6B52 00
6B53 00
6B54 00
6B55 00
6B56 00
6B57 00
6B58 00     4980        .BYTE 0,0,0,0,0,0,0,127
6B59 00
6B5A 00
6B5B 00
6B5C 00
6B5D 00
6B5E 00
6B5F 7F
6B60 7F     4990        .BYTE 127,0,0,0,0,0,0,0
6B61 00
6B62 00
6B63 00
6B64 00
6B65 00
6B66 00
6B67 00
6B68 00     5000        .BYTE 0,1,2,4,3,144,161,0
6B69 01
6B6A 02
6B6B 04
6B6C 03
6B6D 90
6B6E A1
6B6F 00
6B70 00     5010        .BYTE 0,0,145,160,73,0,147,0
6B71 00
6B72 91
6B73 A0
6B74 49
6B75 00
6B76 93
6B77 00
6B78 00     5020        .BYTE 0,152,151,0,164,191,191,168
6B79 98
6B7A 97
6B7B 00
6B7C A4
6B7D BF
6B7E BF
6B7F A8
6B80 B4     5030        .BYTE 180,0,0,0,0,0,0,0
6B81 00
6B82 00
6B83 00
6B84 00
6B85 00
6B86 00
6B87 00
6B88 00     5040        .BYTE 0,0,0,0,0,0,0,127
6B89 00
6B8A 00
6B8B 00
6B8C 00
6B8D 00
6B8E 00
6B8F 7F
6B90 7F     5050        .BYTE 127,0,0,0,0,0,0,0
6B91 00
6B92 00
6B93 00
6B94 00
6B95 00
6B96 00
6B97 00
6B98 00     5060        .BYTE 0,5,3,6,2,1,143,159
6B99 05
6B9A 03
6B9B 06
6B9C 02
6B9D 01
6B9E 8F
6B9F 9F
6BA0 00     5070        .BYTE 0,0,0,146,186,165,187,166
6BA1 00
6BA2 00
6BA3 92
6BA4 BA
6BA5 A5
6BA6 BB
6BA7 A6
6BA8 A7     5080        .BYTE 167,188,182,172,191,191,191,178
6BA9 BC
6BAA B6
6BAB AC
6BAC BF
6BAD BF
6BAE BF
6BAF B2
6BB0 AE     5090        .BYTE 174,0,74,152,154,157,156,159
6BB1 00
6BB2 4A
6BB3 98
6BB4 9A
6BB5 9D
6BB6 9C
6BB7 9F
```

```
6BB8 00     5100        .BYTE 0,0,0,0,0,0,0,127
6BB9 00
6BBA 00
6BBB 00
6BBC 00
6BBD 00
6BBE 00
6BBF 7F
6BC0 7F     5110        .BYTE 127,0,0,4,5,1,5,2
6BC1 00
6BC2 00
6BC3 04
6BC4 05
6BC5 01
6BC6 05
6BC7 02
6BC8 03     5120        .BYTE 3,6,1,4,5,6,2,145
6BC9 06
6BCA 01
6BCB 04
6BCC 05
6BCD 06
6BCE 02
6BCF 91
6BD0 9E     5130        .BYTE 158,0,0,176,170,191,191,191
6BD1 00
6BD2 00
6BD3 B0
6BD4 AA
6BD5 BF
6BD6 BF
6BD7 BF
6BD8 B2     5140        .BYTE 178,173,183,184,184,185,163,181
6BD9 AD
6BDA B7
6BDB B8
6BDC B8
6BDD B9
6BDE A3
6BDF B5
6BE0 99     5150        .BYTE 153,157,155,150,0,0,0,148
6BE1 9D
6BE2 9B
6BE3 96
6BE4 00
6BE5 00
6BE6 00
6BE7 94
6BE8 00     5160        .BYTE 0,0,0,0,0,0,0,127
6BE9 00
6BEA 00
6BEB 00
6BEC 00
6BED 00
6BEE 00
6BEF 7F
6BF0 7F     5170        .BYTE 127,0,0,5,3,6,4,1
6BF1 00
6BF2 00
6BF3 05
6BF4 03
6BF5 06
6BF6 04
6BF7 01
6BF8 04     5180        .BYTE 4,2,0,3,4,1,6,0
6BF9 02
6BFA 00
6BFB 03
6BFC 04
6BFD 01
6BFE 06
6BFF 00
6C00 92     5190        .BYTE 146,186,167,171,191,191,191,191
6C01 BA
6C02 A7
6C03 AB
6C04 BF
6C05 BF
6C06 BF
6C07 BF
6C08 A8     5200        .BYTE 168,180,0,0,176,170,191,169
6C09 B4
6C0A 00
6C0B 00
6C0C B0
6C0D AA
6C0E BF
6C0F A9
6C10 BB     5210        .BYTE 187,166,167,180,0,0,0,0
6C11 A6
6C12 A7
6C13 B4
6C14 00
6C15 00
6C16 00
6C17 00
6C18 00     5220        .BYTE 0,0,0,0,0,0,0,127
6C19 00
6C1A 00
6C1B 00
6C1C 00
6C1D 00
6C1E 00
6C1F 7F
```

```
6C20 7F     5230        .BYTE 127,0,0,0,0,0,0,0
6C21 00
6C22 00
6C23 00
6C24 00
6C25 00
6C26 00
6C27 00
6C28 00     5240        .BYTE 0,0,0,0,0,0,0,0
6C29 00
6C2A 00
6C2B 00
6C2C 00
6C2D 00
6C2E 00
6C2F 00
6C30 B1     5250        .BYTE 177,172,191,191,191,191,191,191
6C31 AC
6C32 BF
6C33 BF
6C34 BF
6C35 BF
6C36 BF
6C37 BF
6C38 BF     5260        .BYTE 191,169,181,175,171,191,191,191
6C39 A9
6C3A B5
6C3B AF
6C3C AB
6C3D BF
6C3E BF
6C3F BF
6C40 BF     5270        .BYTE 191,191,191,169,165,181,5,4
6C41 BF
6C42 BF
6C43 A9
6C44 A5
6C45 B5
6C46 05
6C47 04
6C48 02     5280        .BYTE 2,3,6,1,6,2,1,127
6C49 03
6C4A 06
6C4B 01
6C4C 06
6C4D 02
6C4E 01
6C4F 7F
6C50 7F     5290        .BYTE 127,0,0,0,0,0,0,0
6C51 00
6C52 00
6C53 00
6C54 00
6C55 00
6C56 00
6C57 00
6C58 00     5300        .BYTE 0,0,0,0,0,0,0,0
6C59 00
6C5A 00
6C5B 00
6C5C 00
6C5D 00
6C5E 00
6C5F 00
6C60 A4     5310        .BYTE 164,191,191,191,191,191,191,191
6C61 BF
6C62 BF
6C63 BF
6C64 BF
6C65 BF
6C66 BF
6C67 BF
6C68 BF     5320        .BYTE 191,191,168,172,191,191,191,191
6C69 BF
6C6A A8
6C6B AC
6C6C BF
6C6D BF
6C6E BF
6C6F BF
6C70 BF     5330        .BYTE 191,191,191,191,191,168,166,167
6C71 BF
6C72 BF
6C73 BF
6C74 BF
6C75 A8
6C76 A6
6C77 A7
6C78 B5     5340        .BYTE 181,1,2,3,4,3,3,127
6C79 01
6C7A 02
6C7B 03
6C7C 04
6C7D 03
6C7E 03
6C7F 7F
6C80 7F     5350        .BYTE 127,127,127,127,127,127,127,127
6C81 7F
6C82 7F
6C83 7F
6C84 7F
6C85 7F
6C86 7F
6C87 7F
```

```
6C86 7F     5360        .BYTE 127,127,127,127,127,127,127,127
6C87 7F
6C88 7F
6C89 7F
6C8A 7F
6C8B 7F
6C8C 7F
6C8D 7F
6C8E 7F     5370        .BYTE 127,127,127,127,127,127,127,127
6C8F 7F
6C90 7F
6C91 7F
6C92 7F
6C93 7F
6C94 7F
6C95 7F
6C96 7F     5380        .BYTE 127,127,127,127,127,127,127,127
6C97 7F
6C98 7F
6C99 7F
6C9A 7F
6C9B 7F
6C9C 7F
6C9D 7F
6C9E 7F     5390        .BYTE 127,127,127,127,127,127,127,127
6C9F 7F
6CA0 7F
6CA1 7F
6CA2 7F
6CA3 7F
6CA4 7F
6CA5 7F
6CA6 7F     5400        .BYTE 127,127,127,127,127,127,127,127,127
6CA7 7F
6CA8 7F
6CA9 7F
6CAA 7F
6CAB 7F
6CAC 7F
6CAD 7F
6CAE 7F     5410 STKTAB .BYTE $FF,$FF,$FF,$FF,$FF,$FF,$FF,1
6CAF 7F
6CB0 7F
6CB1 FF
6CB2 FF
6CB3 FF
6CB4 FF
6CB5 FF
6CB6 FF
6CB7 FF     5420        .BYTE $FF,$FF,$FF,3,$FF,2,0,$FF
6CB8 01
6CB9 FF
6CBA FF
6CBB FF
6CBC 03
6CBD FF
6CBE 02
6CBF 00
6CC0 FF
6CC1 28     5430 SSNCOD .BYTE 40,40,40,20,0,0,0,0,0,20,40,40
6CC2 28
6CC3 28
6CC4 14
6CC5 00
6CC6 00
6CC7 00
6CC8 00
6CC9 00
6CCA 14
6CCB 28
6CCC 28
6CCD 06     5440 TRNTAB .BYTE 6,12,8,0,0,18,14,8,20,128
6CCE 0C
6CCF 08
6CD0 00
6CD1 00
6CD2 12
6CD3 0E
6CD4 08
6CD5 14
6CD6 80
6CD7 04     5450        .BYTE 4,8,6,0,0,18,13,6,16,128
6CD8 08
6CD9 06
6CDA 00
6CDB 00
6CDC 12
6CDD 0D
6CDE 06
6CDF 10
6CE0 80
6CE1 18     5460        .BYTE 24,30,24,0,0,30,30,26,28,128
6CE2 1E
6CE3 18
6CE4 00
6CE5 00
6CE6 1E
6CE7 1E
6CE8 1A
6CE9 1C
6CEA 80
6CEB 1E     5470        .BYTE 30,30,30,0,0,30,30,30,30,128
6CEC 1E
6CED 1E
6CEE 00
6CEF 00
```

```
6CF0 1E
6CF1 1E
6CF2 1E
6CF3 1E
6CF4 80
6CF5 0A     5480        .BYTE 10,16,10,12,12,24,28,12,24,128
6CF6 10
6CF7 0A
6CF8 0C
6CF9 0C
6CFA 18
6CFB 1C
6CFC 0C
6CFD 18
6CFE 80
6CFF 06     5490        .BYTE 6,10,8,8,8,24,28,8,20,128
6D00 0A
6D01 08
6D02 08
6D03 08
6D04 18
6D05 1C
6D06 08
6D07 14
6D08 80
6D09 28     5500 BHX1   .BYTE 40,39,38,36,35,34,22,15,15,14
6D0A 27
6D0B 26
6D0C 24
6D0D 23
6D0E 22
6D0F 16
6D10 0F
6D11 0F
6D12 0E
6D13 28     5510        .BYTE 40,39,38,35,35,34,22,15,14,14,19,19
6D14 27
6D15 26
6D16 23
6D17 23
6D18 22
6D19 16
6D1A 0F
6D1B 0E
6D1C 0E
6D1D 13
6D1E 13
6D1F 23     5520 BHY1   .BYTE 35,35,35,33,36,36,4,7,7,8
6D20 23
6D21 23
6D22 21
6D23 24
6D24 24
6D25 04
6D26 07
6D27 07
6D28 08
6D29 24     5530        .BYTE 36,36,36,33,37,37,3,6,7,7,4,3
6D2A 24
6D2B 24
6D2C 21
6D2D 25
6D2E 25
6D2F 03
6D30 06
6D31 07
6D32 07
6D33 04
6D34 03
6D35 28     5540 BHX2   .BYTE 40,39,38,35,35,34,22,15,14,14
6D36 27
6D37 26
6D38 23
6D39 23
6D3A 22
6D3B 16
6D3C 0F
6D3D 0E
6D3E 0E
6D3F 28     5550        .BYTE 40,39,38,36,35,34,22,15,15,14,19,19
6D40 27
6D41 26
6D42 24
6D43 23
6D44 22
6D45 16
6D46 0F
6D47 0F
6D48 0E
6D49 13
6D4A 13
6D4B 24     5560 BHY2   .BYTE 36,36,36,33,37,37,3,6,7,7
6D4C 24
6D4D 24
6D4E 21
6D4F 25
6D50 25
6D51 03
6D52 06
6D53 07
6D54 07
6D55 23     5570        .BYTE 35,35,35,33,36,36,4,7,7,8,3,4
6D56 23
6D57 23
```

```
6D58 21
6D59 24
6D5A 24
6D5B 04
6D5C 07
6D5D 07
6D5E 08
6D5F 03
6D60 04
6D61          5580 EXEC   *=   *+159
6E00          5590        .END
```

```
                10 ;EFT VERSION 1.8I (INTERRUPT) 11/30/81 COPYRIGHT CHRIS CRAWFORD 1
                20 ;
                30 ;Page zero RAM
                40 ;
0014            50 RTCLKL   =    $14
004D            60 ATRACT   =    $4D
004E            70 DRKMSK   =    $4E
004F            80 COLRSH   =    $4F
0000            90          *=   $B0
                0100 ;
                0110 ;These locations are used by the interrupt service routine
                0120 ;
00B0            0130 DLSTPT *=   *+2      Zero page pointer to display list
00B2            0140 MAPLO  *=   *+1
00B3            0150 MAPHI  *=   *+1
00B4            0160 CORPS  *=   *+1      number of unit under window
00B5            0170 CURSXL *=   *+1
00B6            0180 CURSXH *=   *+1
00B7            0190 CURSYL *=   *+1      cursor coordinates on screen (map frame)
00B8            0200 CURSYH *=   *+1
00B9            0210 OFFLO  *=   *+1      How far to offset new LMS value
00BA            0220 OFFHI  *=   *+1
00BB            0230 TEMPI  *=   *+1      An all-purpose temporary register
00BC            0240 CNT1   *=   *+1      DLI counter
00BD            0250 CNT2   *=   *+1      DLI counter for movable map DLI
00BE            0260 CHUNKX *=   *+1      cursor coordinates (pixel frame)
00BF            0270 CHUNKY *=   *+1
                0280 ;
                0290 ;THIS VALUE IS USED BY MAINLINE ROUTINE AND INTERRUPT
                0300 ;
00C9            0310 TURN   =    $C9
                0320 ;
                0330 ;OS locations (see OS manual)
                0340 ;
02C0            0350 PCOLR0 =    $02C0
0278            0360 STICK  =    $0278
02FC            0370 CH     =    $2FC
                0380 ;
                0390 ;HARDWARE LOCATIONS
                0400 ;
D000            0410 HPOSP0 =    $D000
D001            0420 HPOSP1 =    $D001
D002            0430 HPOSP2 =    $D002
D003            0440 HPOSP3 =    $D003
D010            0450 TRIG0  =    $D010
D011            0460 TRIG1  =    $D011
D012            0470 TRIG2  =    $D012
D016            0480 COLPF0 =    $D016
D017            0490 COLPF1 =    $D017
D018            0500 COLPF2 =    $D018
D01A            0510 COLBAK =    $D01A
D01F            0520 CONSOL =    $D01F
D200            0530 AUDF1  =    $D200
D201            0540 AUDC1  =    $D201
D404            0550 HSCROLL=    $D404
D405            0560 VSCROLL=    $D405
D40A            0570 WSYNC  =    $D40A
D409            0580 CHBASE =    $D409
E45C            0590 SETVBV =    $E45C
E462            0600 XITVBV =    $E462
                0610 ;
                0620 ;Page 6 usage
                0630 ;
00C0            0640        *=   $0600
                0650 ;first come locations used by the interrupt service routine
0600            0660 XPOSL  *=   *+1      Horizontal position of
0601            0670 YPOSL  *=   *+1      Vertical position of
0602            0680 YPOSH  *=   *+1      upper left corner of screen window
0603            0690 SCY    *=   *+1      vert position of cursor (player frame)
0604            0700 SHPOS0 *=   *+1      shadows player 0 position
0605            0710 TRCOLR *=   *+1
0606            0720 EARTH  *=   *+1
0607            0730 ICELAT *=   *+1
0608            0740 SEASN1 *=   *+1
0609            0750 SEASN2 *=   *+1
060A            0760 SEASN3 *=   *+1
060B            0770 DAY    *=   *+1
060C            0780 MONTH  *=   *+1
060D            0790 YEAR   *=   *+1
060E            0800 BUTFLG *=   *+1
060F            0810 BUTMSK *=   *+1
0610            0820 TYL    *=   *+1
0611            0830 TYH    *=   *+1
0612            0840 DELAY  *=   *+1      acceleration delay on scrolling
0613            0850 TIMSCL *=   *+1      frame to scroll in
0614            0860 TEMPLO *=   *+1      temporary
0615            0870 TEMPHI *=   *+1
0616            0880 BASEX  *=   *+1      start position for arrow (player frame)
0617            0890 BASEY  *=   *+1
0618            0900 STEPX  *=   *+1      intermediate position of arrow
0619            0910 STEPY  *=   *+1
061A            0920 STPCNT *=   *+1      which intermediate steps arrow is on
061B            0930 ORDCNT *=   *+1      which order arrow is showing
061C            0940 ORD1   *=   *+1      orders record
061D            0950 ORD2   *=   *+1
061E            0960 ARRNDX *=   *+1      arrow index
061F            0970 HOWMNY *=   *+1      how many orders for unit under cursor
0620            0980 KRZX   *=   *+1      maltakreuze coords (player frame)
0621            0990 KRZY   *=   *+1
0622            1000 DBTIMR *=   *+1      joystick debounce timer
0623            1010 STICKI *=   *+1      coded value of stick direction (0-3)
0624            1020 ERRFLG *=   *+1
0625            1030 KRZFLG *=   *+1
```

```
0626          1040 STKFLG *=    *+1
0627          1050 HITFLG *=    *+1
0628          1060 TXL    *=    *+1        temporary values---slightly shifted
0629          1070 TXH    *=    *+1
068F          1080 HANDCP =     $68F
              1090 ;
062A          1100        *=    $5200
5200          1110 PLYR0  *=    *+128
5280          1120 PLYR1  *=    *+128
5300          1130 PLYR2  *=    *+128
5380          1140 PLYR3  *=    *+128
54U0          1150 CORPSX *=    *+159      x-coords of all units (pixel frame)
549F          1160 CORPSY *=    *+159      y-coords of all units (pixel frame)
553E          1170 MSTRNG *=    *+159      muster strengths
55DD          1180 CSTRNG *=    *+159      combat strengths
567C          1190 SWAP   *=    *+159      terrain code underneath unit
571B          1200 ARRIVE *=    *+159      turn of arrival
57BA          1210 WORDS  *=    *+272      various words for messages
58CA          1220 CORPT  *=    *+159      codes for unit types
5969          1230 CORPNO *=    *+159      ID numbers of units
5A08          1240 HDIGIT *=    *+256      tables for displaying numbers (hundreds)
5B08          1250 TDIGIT *=    *+256      tens tables
5C08          1260 ODIGIT *=    *+256      ones tables
5D08          1270 TXTTBL *=    *+96       more text
5D68          1280 MONLEN *=    *+13       table of month lengths
5D75          1290 HMORDS *=    *+159      how many orders each unit has in queue
5E14          1300 WHORDS *=    *+159      what the orders are
5EB3          1310 WHORDH *=    *+159
5F52          1320 BEEPTB *=    *+4        table of beep tones
5F56          1330 ERRMSG *=    *+128      table of error messages
5FD6          1340 XOFF   *=    *+4        offsets for moving maltakreuze
5FDA          1350 YOFF   *=    *+4
5FDE          1360 MASKO  *=    *+4        mask values for decoding orders
5FE2          1370 XADD   *=    *+4        offsets for moving arrow
5FE6          1380 YADD   *=    *+4
5FEA          1390 TRTAB  *=    *+13
5FF7          1400 MLTKRZ *=    *+8        maltese cross shape
              1410 ;
              1420 ;RAM from $6000 to $6430 is taken up by
              1430 ;character sets and the display list
              1440 ;
5FFF          1450        *=    $6431
6431          1460 ARRTAB *=    *+32       arrow shapes
6451          1470        *=    $6450
6450          1480 TXTWDW *=    $6CB1
              1490 ;
6CB1          1500 STKTAB *=    *+16       a joystick decoding table
6CC1          1510 SSNCOD *=    *+12
6CCD          1520 TRNTAB *=    *+60
6D09          1530 BHX1   *=    *+22
6D1F          154U BHY1   *=    *+22
6D35          1550 BHX2   *=    *+22
6D4B          1560 BHY2   *=    *+22
6D61          1570 EXEC   *=    *+159
              1580 ;
              1590 ;everything in here is taken up by the map data
              1600 ;
              1610 ;
              1620 ;This is the vertical blank interrupt routine
              1630 ;It reads the joystick and scrolls the screen
              1640 ;
6E00          1650        *=    $7400
7400 AD11D0   1660        LDA   TRIG1      check for break button
7403 D00F     1670        BNE   Z30        no, check next
7405 A03E     1680        LDY   #62        reset 60 Hertz vector
7407 A2E9     1690        LDX   #233
7409 A907     1700        LDA   #7
740B 205CE4   1710        JSR   SETVBV
740E 68       1720        PLA              reset stack
740F 68       1730        PLA
7410 68       1740        PLA
7411 4C1072   1750        JMP   $7210      break routine
7414 AD8F06   1760 Z30    LDA   HANDCP
7417 F02D     1770        BEQ   A31
7419 AD10D0   1780        LDA   TRIG0
741C F028     1790        BEQ   A31
741E A908     1800        LDA   #$08
7420 8D1FD0   1810        STA   CONSOL
7423 AD1FD0   1820        LDA   CONSOL
7426 2904     1830        AND   #$04
7428 D01C     1840        BNE   A31
742A 8D8F06   1850        STA   HANDCP
742D A930     1860        LDA   #$30
742F 8D7A7B   1870        STA   $7B7A      my trademark
7432 A236     1880        LDX   #$36
7434 BD3E55   1890 LOOPJ  LDA   MSTRNG,X
7437 85BB     1900        STA   TEMP1
7439 4A       1910        LSR   A
743A 65BB     1920        ADC   TEMP1
743C 9002     1930        BCC   A22
743E A9FF     1940        LDA   #$FF
744U 9D3E55   1950 A22    STA   MSTRNG,X
7443 CA       1960        DEX
7444 D0EE     1970        BNE   LOOPJ
              1980 ;
              1990 ;
7446 AD10D0   2000 A31    LDA   TRIG0      button status
7449 0D0F06   2010        ORA   BUTMSK     button allowed?
744C F03B     2020        BEQ   X17
744E AD0E06   2030        LDA   BUTFLG     no button now; previous status
7451 D003     2040        BNE   X23
7453 4CCF77   2050        JMP   NOBUT
7456 A958     2060 X23    LDA   #$58      button just released
7458 8DC002   2070        STA   PCOLR0
```

this code masked out by brute force in final version.
C7400< A9, FF, EA

```
745B A900    2080         LDA  #$00
745D 8D0E06  2090         STA  BUTFLG
7460 8D2506  2100         STA  KRZFLG
7463 8D01D2  2110         STA  AUDC1
7466 A252    2120         LDX  #$52
7468 9D5864  2130 LOOP8   STA  TXTWDW+8,X clear text window
746B CA      2140         DEX
746C 10FA    2150         BPL  LOOP8
746E A908    2160         LDA  #$08
7470 8D1206  2170         STA  DELAY
7473 18      2180         CLC
7474 6514    2190         ADC  RTCLKL
7476 8D1306  2200         STA  TIMSCL
7479 20EF79  2210         JSR  SWITCH
747C A900    2220         LDA  #$00
747E 85B4    2230         STA  CORPS
7480 20357A  2240         JSR  CLRP1
7483 204A7A  2250         JSR  CLRP2
7486 4C6F79  2260         JMP  ENDISR
7489 854D    2270 X17     STA  ATRACT    button is pressed
748B AD7802  2280         LDA  STICK
748E 290F    2290         AND  #$0F
7490 490F    2300         EOR  #$0F
7492 F003    2310         BEQ  X20       joystick active?
7494 4CDA76  2320         JMP  ORDERS    yes
7497 8D2206  2330 X20     STA  DBTIMR    no, set debounce
749A 8D01D2  2340         STA  AUDC1
749D 8D2606  2350         STA  STKFLG
74A0 AD0E06  2360         LDA  BUTFLG
74A3 D003    2370         BNE  BUTHLD    is this the first button pass
74A5 4CAA75  2380         JMP  FBUTPS    yes
74A8 205E7A  2390 BUTHLD  JSR  ERRCLR    no, clear errors
74AB AD2706  2400 X61     LDA  HITFLG
74AE F003    2410         BEQ  X63       anybody in the window?
74B0 4C6F79  2420         JMP  ENDISR    no
74B3 ADFC02  2430 X63     LDA  CH
74B6 C921    2440         CMP  #$21
74B8 D02B    2450         BNE  X80       space bar pressed?
74BA A6B4    2460         LDX  CORPS     yes, check for Russian
74BC E037    2470         CPX  #$37
74BE B025    2480         BCS  X80
74C0 A900    2490         LDA  #$00
74C2 8DFC02  2500         STA  CH
74C5 9D755D  2510         STA  HMORDS,X  clear out orders
74C8 8D1F06  2520         STA  HOWMNY
74CB 8D1A06  2530         STA  STPCNT
74CE A901    2540         LDA  #$01
74D0 8D1B06  2550         STA  ORDCNT
74D3 20357A  2560         JSR  CLRP1
74D6 204A7A  2570         JSR  CLRP2
74D9 AD1606  2580         LDA  BASEX
74DC 8D1806  2590         STA  STEPX
```

```
74DF AD1706  2600         LDA  BASEY
74E2 8D1906  2610         STA  STEPY
74E5 A514    2620 X80     LDA  RTCLKL
74E7 2903    2630         AND  #$03
74E9 F003    2640         BEQ  X54       time to move arrow?
74EB 4C6F79  2650         JMP  ENDISR    no
74EE AC1F06  2660 X54     LDY  HOWMNY    yes
74F1 D003    2670         BNE  X65       any orders to show?
74F3 4C6F75  2680         JMP  PCURSE    no, go ahead to maltakreuze
74F6 20357A  2690 X65     JSR  CLRP1     yes, clear old arrow
74F9 AD1B06  2700         LDA  ORDCNT
74FC A200    2710         LDX  #$00      assume first byte
74FE C905    2720         CMP  #$05
7500 9001    2730         BCC  X52       second byte or first?
7502 E8      2740         INX            second byte
7503 2903    2750 X52     AND  #$03      isolate bit pair index
7505 A8      2760         TAY
7506 B9747A  2770         LDA  BITTAB,Y  get mask
7509 3D1C06  2780 X50     AND  ORD1,X    get orders
             2790 ;
             2800 ;right justify orders
             2810 ;
750C 88      2820         DEY
750D 1002    2830         BPL  X51
750F A003    2840         LDY  #$03
7511 F005    2850 X51     BEQ  X53
7513 4A      2860 LOOP21  LSR  A
7514 4A      2870         LSR  A
7515 88      2880         DEY
7516 D0FB    2890         BNE  LOOP21
7518 8D1E06  2900 X53     STA  ARRNDX
751B 0A      2910         ASL  A
751C 0A      2920         ASL  A
751D 0A      2930         ASL  A
             2940 ;get arrow image and store it to player RAM
751E AA      2950         TAX
751F AC1906  2960         LDY  STEPY
7522 BD3164  2970 X55     LDA  ARRTAB,X
7525 C080    2980         CPY  #$80
7527 B003    2990         BCS  X43
7529 998052  3000         STA  PLYR1,Y
752C E8      3010 X43     INX
752D C8      3020         INY
752E 8A      3030         TXA
752F 2907    3040         AND  #$07
7531 D0EF    3050         BNE  X55
             3060 ;
7533 AD1806  3070         LDA  STEPX     position arrow
7536 8D01D0  3080         STA  HPOSP1
             3090 ;
             3100 ;now step arrow
             3110 ;
```

```
7539 AE1E06 3120         LDX  ARRNDX
753C AD1806 3130         LDA  STEPX
753F 18     3140         CLC
7540 7DE25F 3150         ADC  XADD,X
7543 8D1806 3160         STA  STEPX
7546 AD1906 3170         LDA  STEPY
7549 18     3180         CLC
754A 7DE65F 3190         ADC  YADD,X
754D 8D1906 3200         STA  STEPY
            3210 ;
7550 EE1A06 3220         INC  STPCNT    next step
7553 AD1A06 3230         LDA  STPCNT
7556 2907   3240         AND  #$07
7558 D04D   3250         BNE  X59       If not done end ISR
755A 8D1A06 3260         STA  STPCNT    end of steps
755D EE1B06 3270         INC  ORDCNT    next order
7560 AD1B06 3280         LDA  ORDCNT
7563 CD1F06 3290         CMP  HOWMNY    last order?
7566 903F   3300         BCC  X59       no, out
7568 F03D   3310         BEQ  X59       no, out
756A A901   3320         LDA  #$01
756C 8D1B06 3330         STA  ORDCNT    yes, reset to start of arrow's path
            3340 ;
            3350 ;display maltese cross ('maltakreuze' or KRZ)
            3360 ;
756F AC1906 3370 PCURSE  LDY  STEPY
7572 8C2106 3380         STY  KRZY
7575 A9FF   3390         LDA  #$FF
7577 8D2506 3400         STA  KRZFLG
757A A200   3410         LDX  #$00
757C BDF75F 3420 LOOP24  LDA  MLTKRZ,X
757F C080   3430         CPY  #$80
7581 B003   3440         BCS  X44
7583 990053 3450         STA  PLYR2,Y
7586 C8     3460 X44     INY
7587 E8     3470         INX
7588 E008   3480         CPX  #$08
758A D0F0   3490         BNE  LOOP24
758C AD1806 3500         LDA  STEPX
758F 38     3510         SEC
7590 E901   3520         SBC  #$01
7592 8D2006 3530         STA  KRZX
7595 8D02D0 3540         STA  HPOSP2
7598 20557A 3550         JSR  CLRP1     clear arrow
759B AD1606 3560         LDA  BASEX     reset arrow's coords
759E 8D1806 3570         STA  STEPX
75A1 AD1706 3580         LDA  BASEY
75A4 8D1906 3590         STA  STEPY
            3600 ;
V4C6F79 3610 X59     JMP  ENDISR
            3620 ;
            3630 ;FIRST BUTTON PASS
            3640 ;looks for a unit inside cursor
            3650 ;if there is one, puts unit info into text window
            3660 ;
75AA A9FF   3670 FBUTPS  LDA  #$FF
75AC 8D0E06 3680         STA  BUTFLG
            3690 ;
            3700 ;first get coords of center of cursor (map frame)
            3710 ;
75AF A5B5   3720 X24     LDA  CURSXL
75B1 18     3730         CLC
75B2 6906   3740         ADC  #$06
75B4 8D2806 3750         STA  TXL
75B7 A5B6   3760         LDA  CURSXH
75B9 6900   3770         ADC  #$00
75BB 8D2906 3780         STA  TXH
75BE A5B7   3790         LDA  CURSYL
75C0 18     3800         CLC
75C1 6909   3810         ADC  #$09
75C3 8D1006 3820         STA  TYL
75C6 A5B8   3830         LDA  CURSYH
75C8 6900   3840         ADC  #$00
75CA 8D1106 3850         STA  TYH
75CD AD2906 3860         LDA  TXH
75D0 4A     3870         LSR  A
75D1 AD2806 3880         LDA  TXL
75D4 6A     3890         ROR  A
75D5 4A     3900         LSR  A
75D6 4A     3910         LSR  A
            3920 ;
            3930 ;coords of cursor (pixel frame)
            3940 ;
75D7 85BE   3950         STA  CHUNKX
75D9 AD1106 3960         LDA  TYH
75DC 4A     3970         LSR  A
75DD AA     3980         TAX
75DE AD1006 3990         LDA  TYL
75E1 6A     4000         ROR  A
75E2 A8     4010         TAY
75E3 8A     4020         TXA
75E4 4A     4030         LSR  A
75E5 98     4040         TYA
75E6 6A     4050         ROR  A
75E7 4A     4060         LSR  A
75E8 4A     4070         LSR  A
75E9 85BF   4080         STA  CHUNKY
            4090 ;
            4100 ;now look for a match with unit coordinates
            4110 ;
75EB A29E   4120         LDX  #$9E
75ED DD9F54 4130 LOOP6   CMP  CORPSY,X
75F0 F00C   4140         BEQ  MAYBE
75F2 CA     4150 X16     DEX
```

```
75F3 D0F8   4160         BNE  LOOP6
75F5 86B4   4170         STX  CORPS      no match obtained
75F7 CA     4180         DEX
75F8 8E2706 4190         STX  HITFLG
75FB 4C6F79 4200         JMP  ENDISR
            4210 ;
75FE A5BE   4220 MAYBE   LDA  CHUNKX
7600 DD0054 4230         CMP  CORPSX,X
7603 D00B   4240         BNE  X35
7605 BD1B57 4250         LDA  ARRIVE,X
7608 3006   4260         BMI  X35
760A C5C9   4270         CMP  TURN
760C 9007   4280         BCC  MATCH
760E F005   4290         BEQ  MATCH
7610 A5BF   4300 X35     LDA  CHUNKY
7612 4CF275 4310         JMP  X16
            4320 ;
            4330 ;match obtained
            4340 ;
7615 A900   4350 MATCH   LDA  #$00
7617 8D2706 4360         STA  HITFLG     note match
761A 8DFC02 4370         STA  CH
761D A95C   4380         LDA  #$5C
761F 8DC002 4390         STA  PCOLR0     light up cursor
            4400 ;
            4410 ;display unit specs
            4420 ;
7622 86B4   4430         STX  CORPS
7624 A00D   4440         LDY  #$0D
7626 BD6959 4450         LDA  CORPNO,X   ID number
7629 20B27B 4460         JSR  DNUMBR
762C C8     4470         INY
762D A6B4   4480         LDX  CORPS
762F BDCA58 4490         LDA  CORPT,X    first name
7632 85BB   4500         STA  TEMP1
7634 29F0   4510         AND  #$F0
7636 4A     4520         LSR  A
7637 20DA79 4530         JSR  ENTRY2
763A A5BB   4540         LDA  TEMP1
763C 290F   4550         AND  #$0F       second name
763E 18     4560         CLC
763F 6908   4570         ADC  #$08
7641 20C079 4580         JSR  DWORDS
7644 A91E   4590         LDA  #$1E
7646 A6B4   4600         LDX  CORPS
7648 E037   4610         CPX  #$37
764A B002   4620         BCS  X26
764C A91D   4630         LDA  #$1D
764E 20C079 4640 X26     JSR  DWORDS     display unit size (corps or army)
7651 A038   4650         LDY  #$38
7653 A91F   4660         LDA  #$1F       "MUSTER"
7655 20C079 4670         JSR  DWORDS
7658 88     4680         DEY
7659 A91A   4690         LDA  #$1A       ":"
765B 995064 4700         STA  TXTWDW,Y
765E C8     4710         INY
765F C8     4720         INY
7660 A6B4   4730         LDX  CORPS
7662 BD3E55 4740         LDA  MSTRNG,X   muster strength
7665 20B27B 4750         JSR  DNUMBR
7668 C8     4760         INY
7669 C8     4770         INY
766A A920   4780         LDA  #$20       "COMBAT"
766C 20C079 4790         JSR  DWORDS
766F A921   4800         LDA  #$21       "STRENGTH"
7671 20C079 4810         JSR  DWORDS
7674 88     4820         DEY
7675 A91A   4830         LDA  #$1A       ":"
7677 995064 4840         STA  TXTWDW,Y
767A C8     4850         INY
767B C8     4860         INY
767C A6B4   4870         LDX  CORPS
767E BDDD55 4880         LDA  CSTRNG,X   combat strength
7681 20B27B 4890         JSR  DNUMBR
7684 20EF79 4900 X27     JSR  SWITCH     flip unit with terrain
7687 A5B4   4910         LDA  CORPS
7689 C937   4920         CMP  #$37
768B 9007   4930         BCC  X79        Russian?
768D A9FF   4940         LDA  #$FF       yes, mask orders and exit
768F 8D2706 4950         STA  HITFLG
7692 3043   4960         BMI  X75
            4970 ;
            4980 ;German unit
            4990 ;set up orders display
            5000 ;first calculate starting coords (BASEX, BASEY)
            5010 ;
7694 A901   5020 X79     LDA  #$01
7696 8D1B06 5030         STA  ORDCNT
7699 A900.  5040         LDA  #$00
769B 8D1A06 5050         STA  STPCNT
            5060 ;
769E AD2806 5070         LDA  TXL
76A1 2907   5080         AND  #$07
76A3 18     5090         CLC
76A4 6901   5100         ADC  #$01
76A6 18     5110         CLC
76A7 6D0406 5120         ADC  SHPOS0
76AA 8D1606 5130         STA  BASEX
76AD 8D1806 5140         STA  STEPX
            5150 ;
76B0 AD1006 5160         LDA  TYL
76B3 290F   5170         AND  #$0F
76B5 4A     5180         LSR  A
76B6 38     5190         SEC
```

```
76B7 E901   5200         SBC  #$01
76B9 18     5210         CLC
76BA 6D0306 5220         ADC  SCY
76BD 8D1706 5230         STA  BASEY
76C0 8D1906 5240         STA  STEPY
            5250 ;
            5260 ;now set up page 6 values
            5270 ;
76C3 A6B4   5280         LDX  CORPS
76C5 BD755D 5290         LDA  HMORDS,X
76C8 8D1F06 5300         STA  HOWMNY
76CB BD145E 5310         LDA  WHORDS,X
76CE 8D1C06 5320         STA  ORD1
76D1 BDB35E 5330         LDA  WHORDH,X
76D4 8D1D06 5340         STA  ORD2
76D7 4C6F79 5350 X75     JMP  ENDISR
            5360 ;
            5370 ;ORDERS INPUT ROUTINE
            5380 ;
76DA AD2606 5390 ORDERS  LDA  STKFLG
76DD D0F8   5400         BNE  X75
76DF A6B4   5410         LDX  CORPS
76E1 E037   5420         CPX  #$37
76E3 9005   5430         BCC  X64        Russian?
76E5 A200   5440         LDX  #$00       yes, error
76E7 4CAC77 5450         JMP  SQUAWK
76EA BD755D 5460 X64     LDA  HMORDS,X
76ED C908   5470         CMP  #$08
76EF 9005   5480         BCC  X66        only 8 orders allowed
76F1 A220   5490         LDX  #$20
76F3 4CAC77 5500         JMP  SQUAWK
76F6 AD2506 5510 X66     LDA  KRZFLG
76F9 D005   5520         BNE  X67        must wait for maltakreuze
76FB A240   5530         LDX  #$40
76FD 4CAC77 5540         JMP  SQUAWK
7700 EE2206 5550 X67     INC  DBTIMR
7703 AD2206 5560         LDA  DBTIMR     wait for debounce time
7706 C910   5570         CMP  #$10
7708 B002   5580         BCS  X68
770A 90CB   5590         BCC  X75
770C A900   5600 X68     LDA  #$00
770E 8D2206 5610         STA  DBTIMR     reset debounce timer
7711 AE7802 5620         LDX  STICK
7714 BDB16C 5630         LDA  STKTAB,X
7717 1005   5640         BPL  X69
7719 A260   5650         LDX  #$60       no diagonal orders allowed
771B 4CAC77 5660         JMP  SQUAWK
            5670 ;
            5680 ;OK, this is a good order
            5690 ;
771E A8     5700 X69     TAY
771F 8D2306 5710         STA  STICK1
```

```
7722 B9525F 5720         LDA  BEEPTB,Y
7725 8D00D2 5730         STA  AUDF1      "BEEP!"
7728 A9A8   5740         LDA  #$A8
772A 8D01D2 5750         STA  AUDC1
772D A9FF   5760         LDA  #$FF
772F 8D2606 5770         STA  STKFLG
            5780 ;
7732 A6B4   5790         LDX  CORPS
7734 FE755D 5800         INC  HMORDS,X
7737 BD755D 5810         LDA  HMORDS,X
773A 8D1F06 5820         STA  HOWMNY
773D 38     5830         SEC
773E E901   5840         SBC  #$01
7740 2903   5850         AND  #$03
7742 A8     5860         TAY
7743 84BB   5870         STY  TEMP1
7745 BD755D 5880         LDA  HMORDS,X
7748 38     5890         SEC
7749 E901   5900         SBC  #$01
774B 4A     5910         LSR  A
774C 4A     5920         LSR  A
774D AA     5930         TAX
774E AD2306 5940         LDA  STICK1
            5950 ;isolate order
7751 88     5960 X71     DEY
7752 3005   5970         BMI  X70
7754 0A     5980         ASL  A
7755 0A     5990         ASL  A
7756 4C5177 6000         JMP  X71
7759 A4BB   6010 X70     LDY  TEMP1
775B 5D1C06 6020         EOR  ORD1,X     fold in new order (sneaky code)
775E 39DE5F 6030         AND  MASKO,Y
7761 5D1C06 6040         EOR  ORD1,X
7764 9D1C06 6050         STA  ORD1,X
7767 AD1C06 6060         LDA  ORD1
776A A6B4   6070         LDX  CORPS
776C 9D145E 6080         STA  WHORDS,X
776F AD1D06 6090         LDA  ORD2
7772 9DB35E 6100         STA  WHORDH,X
            6110 ;
            6120 ;move maltakreuze
            6130 ;
7775 204A7A 6140         JSR  CLRP2
7778 AE2306 6150         LDX  STICK1
777B AD2006 6160         LDA  KRZX
777E 18     6170         CLC
777F 7DD65F 6180         ADC  XOFF,X
7782 8D2006 6190         STA  KRZX
7785 AD2106 6200         LDA  KRZY
7788 18     6210         CLC
7789 7DDA5F 6220         ADC  YOFF,X
778C 8D2106 6230         STA  KRZY
```

```
778F AD2006 6240 DSPKRZ LDA  KRZX      display it
7792 8D02D0 6250        STA  HPOSP2
7795 AC2106 6260        LDY  KRZY
7798 A200   6270        LDX  #$00
779A BDF75F 6280 LOOP26 LDA  MLTKRZ,X
779D C080   6290        CPY  #$80
779F B003   6300        BCS  X45
77A1 990053 6310        STA  PLYR2,Y
77A4 C8     6320 X45    INY
77A5 E8     6330        INX
77A6 E008   6340        CPX  #$08
77A8 D0F0   6350        BNE  LOOP26
77AA F043   6360        BEQ  EXITI
            6370 ;
            6380 ;ERROR on inputs routine
            6390 ;squawks speaker and puts out error message
            6400 ;
77AC A069   6410 SQUAWK LDY  #$69
77AE BD565F 6420 LOOP28 LDA  ERRMSG,X
77B1 38     6430        SEC
77B2 E920   6440        SBC  #$20
77B4 995064 6450        STA  TXTWDW,Y
77B7 C8     6460        INY
77B8 E8     6470        INX
77B9 8A     6480        TXA
77BA 291F   6490        AND  #$1F
77BC D0F0   6500        BNE  LOOP28
77BE A968   6510        LDA  #$68
77C0 8D01D2 6520        STA  AUDC1
77C3 A950   6530        LDA  #$50
77C5 8D00D2 6540        STA  AUDF1     "HONKI"
77C8 A9FF   6550        LDA  #$FF
77CA 8D2406 6560        STA  ERRFLG
77CD 3020   6570        BMI  EXITI
            6580 ;
            6590 ;NO BUTTON PRESSED ROUTINE
            6600 ;
77CF 8D2206 6610 NOBUT  STA  DBTIMR
77D2 AD7802 6620        LDA  STICK
77D5 290F   6630        AND  #$0F
77D7 490F   6640        EOR  #$0F
77D9 D017   6650        BNE  SCROLL
77DB 8D01D2 6660        STA  AUDC1
77DE 8D2606 6670        STA  STKFLG
77E1 A908   6680        LDA  #$08
77E3 8D1206 6690        STA  DELAY
77E6 18     6700        CLC
77E7 6514   6710        ADC  RTCLKL
77E9 8D1306 6720        STA  TIMSCL
77EC 205E7A 6730        JSR  ERRCLR
77EF 4C6F79 6740 EXITI  JMP  ENDISR
77F2 A900   6750 SCROLL LDA  #$00
77F4 854D   6760        STA  ATRACT
            6770 ;
            6780 ;acceleration feature of cursor
            6790 ;
77F6 AD1306 6800        LDA  TIMSCL
77F9 C514   6810        CMP  RTCLKL
77FB D0F2   6820        BNE  EXITI
77FD AD1206 6830        LDA  DELAY
7800 C901   6840        CMP  #$01
7802 F006   6850        BEQ  X21
7804 38     6860        SEC
7805 E901   6870        SBC  #$01
7807 8D1206 6880        STA  DELAY
780A 18     6890 X21    CLC
780B 6514   6900        ADC  RTCLKL
780D 8D1306 6910        STA  TIMSCL
            6920 ;
7810 A900   6930        LDA  #$00
7812 85B9   6940        STA  OFFLO
7814 85BA   6950        STA  OFFHI     zero the offset
            6960 ;
7816 AD7802 6970        LDA  STICK     get joystick reading
7819 48     6980        PHA            save it on stack for other bit checks
781A 2908   6990        AND  #$08      joystick left?
781C D03A   7000        BNE  CHKRT     no, move on
781E A5B5   7010        LDA  CURSXL
7820 D004   7020        BNE  X13
7822 A6B6   7030        LDX  CURSXH
7824 F071   7040        BEQ  CHKUP
7826 38     7050 X13    SEC
7827 E901   7060        SBC  #$01
7829 85B5   7070        STA  CURSXL
782B B002   7080        BCS  X14
782D C6B6   7090        DEC  CURSXH
782F AD0406 7100 X14    LDA  SHPOS0
7832 C9BA   7110        CMP  #$BA
7834 F00B   7120        BEQ  X1
7836 18     7130        CLC
7837 6901   7140        ADC  #$01
7839 8D0406 7150        STA  SHPOS0
783C 8D00D0 7160        STA  HPOSP0
783F D056   7170        BNE  CHKUP
7841 AD0006 7180 X1     LDA  XPOSL
7844 38     7190        SEC            decrement x-coordinate
7845 E901   7200        SBC  #$01
7847 8D0006 7210        STA  XPOSL
784A 2907   7220        AND  #$07
784C 8D04D4 7230        STA  HSCROLL   fine scroll
784F C907   7240        CMP  #$07      scroll overflow?
7851 D044   7250        BNE  CHKUP     no, move on
7853 E6B9   7260        INC  OFFLO     yes, mark it for offset
7855 B8     7270        CLV
```

```
7856 503F   7280          BVC  CHKUP     no point in checking for joystick right
7858 68     7290 CHKRT    PLA            get back joystick byte
7859 48     7300          PHA            save it again
785A 2904   7310          AND  #$04      joystick right?
785C D039   7320          BNE  CHKUP     no, move on
785E A5B5   7330          LDA  CURSXL
7860 C964   7340          CMP  #$64
7862 D004   7350          BNE  X12
7864 A6B6   7360          LDX  CURSXH
7866 D02F   7370          BNE  CHKUP
7868 18     7380 X12      CLC
7869 6901   7390          ADC  #$01
786B 85B5   7400          STA  CURSXL
786D 9002   7410          BCC  X15
786F E6B6   7420          INC  CURSXH
7871 AD0406 7430 X15      LDA  SHPOS0
7874 C936   7440          CMP  #$36
7876 F00B   7450          BEQ  X2
7878 38     7460          SEC
7879 E901   7470          SBC  #$01
787B 8D0406 7480          STA  SHPOS0
787E 8D00D0 7490          STA  HPOSP0
7881 D014   7500          BNE  CHKUP
7883 AD0006 7510 X2       LDA  XPOSL
7886 18     7520          CLC            no, increment x-coordinate
7887 6901   7530          ADC  #$01
7889 8D0006 7540          STA  XPOSL
788C 2907   7550 X4       AND  #$07
788E 8D04D4 7560          STA  HSCROLL   fine scroll
7891 D004   7570   PBNE  CHKUP      scroll overflow? if not, move on
7893 C6B9   7580          DEC  OFFLO     yes, set up offset for character scroll
7895 C6BA   7590          DEC  OFFHI
7897 68     7600 CHKUP    PLA            joystick up?
7898 4A     7610          LSR  A
7899 48     7620          PHA
789A B05A   7630          BCS  CHKDN     no, ramble on
789C A5B7   7640          LDA  CURSYL
789E C95E   7650          CMP  #$5E
78A0 D006   7660          BNE  X3
78A2 A6B8   7670          LDX  CURSYH
78A4 E002   7680          CPX  #$02
78A6 F04E   7690          BEQ  CHKDN
78A8 E6B7   7700 X3       INC  CURSYL
78AA D002   7710          BNE  X11
78AC E6B8   7720          INC  CURSYH
78AE AE0306 7730 X11      LDX  SCY
78B1 E01B   7740          CPX  #$1B
78B3 F01D   7750          BEQ  X6
78B5 E6B7   7760          INC  CURSYL
78B7 D002   7770          BNE  X18
78B9 E6B8   7780          INC  CURSYH
78BB CA     7790 X18      DEX
78BC 8E0306 7800          STX  SCY
78BF 8A     7810          TXA
78C0 18     7820          CLC
78C1 6912   7830          ADC  #$12
78C3 85BB   7840          STA  TEMP1
78C5 BD0052 7850 LOOP4    LDA  PLYR0,X
78C8 9DFF51 7860          STA  PLYR0-1,X
78CB E8     7870          INX
78CC E4BB   7880          CPX  TEMP1
78CE D0F5   7890          BNE  LOOP4
78D0 F024   7900          BEQ  CHKDN
78D2 AD0106 7910 X6       LDA  YPOSL
78D5 38     7920          SEC
78D6 E901   7930          SBC  #$01
78D8 B003   7940          BCS  X7
78DA CE0206 7950          DEC  YPOSH
78DD 8D0106 7960 X7       STA  YPOSL
78E0 290F   7970          AND  #$0F
78E2 8D05D4 7980          STA  VSCROLL   fine scroll
78E5 C90F   7990          CMP  #$0F
78E7 D00D   8000          BNE  CHKDN     scroll overflow? if not, amble on
78E9 A5B9   8010          LDA  OFFLO     yes, set up offset for character scroll
78EB 38     8020          SEC
78EC E930   8030          SBC  #$30
78EE 85B9   8040          STA  OFFLO
78F0 A5BA   8050          LDA  OFFHI
78F2 E900   8060          SBC  #$00
78F4 85BA   8070          STA  OFFHI
78F6 68     8080 CHKDN    PLA            joystick down?
78F7 4A     8090          LSR  A
78F8 B05F   8100          BCS  CHGDL     no, trudge on
78FA A5B7   8110          LDA  CURSYL
78FC C902   8120          CMP  #$02
78FE D004   8130          BNE  X5
7900 A6B8   8140          LDX  CURSYH
7902 F055   8150          BEQ  CHGDL
7904 38     8160 X5       SEC
7905 E901   8170          SBC  #$01
7907 85B7   8180          STA  CURSYL
7909 B002   8190          BCS  X10
790B C6B8   8200          DEC  CURSYH
790D AE0306 8210 X10      LDX  SCY
7910 E04E   8220          CPX  #$4E
7912 F023   8230          BEQ  X8
7914 38     8240          SEC
7915 E901   8250          SBC  #$01
7917 85B7   8260          STA  CURSYL
7919 B002   8270          BCS  X19
791B C6B8   8280          DEC  CURSYH
791D E8     8290 X19      INX
791E 8E0306 8300          STX  SCY
7921 8A     8310          TXA
```

```
7922 18      8320         CLC
7923 6912    8330         ADC  #$12
7925 CA      8340         DEX
7926 CA      8350         DEX
7927 86BB    8360         STX  TEMP1
7929 AA      8370         TAX
792A BDFF51  8380 LOOP5   LDA  PLYR0-1,X
792D 9D0052  8390         STA  PLYR0,X
7930 CA      8400         DEX
7931 E4BB    8410         CPX  TEMP1
7933 D0F5    8420         BNE  LOOP5
7935 F022    8430         BEQ  CHGDL
7937 AD0106  8440 X8      LDA  YPOSL
793A 18      8450         CLC             no, decrement y-coordinate
793B 6901    8460         ADC  #$01
793D 8D0106  8470         STA  YPOSL
7940 9003    8480         BCC  X9
7942 EE0206  8490         INC  YPOSH
7945 290F    8500 X9      AND  #$0F
7947 8D05D4  8510         STA  VSCROLL    fine scroll
794A D00D    8520         BNE  CHGDL      no, move on
794C A5B9    8530         LDA  OFFLO      yes, mark offset
794E 18      8540         CLC
794F 6930    8550         ADC  #$30
7951 85B9    8560         STA  OFFLO
7953 A5BA    8570         LDA  OFFHI
7955 6900    8580         ADC  #$00
7957 85BA    8590         STA  OFFHI
             8600 ;
             8610 ;In this loop we add the offset values to the existing
             8620 ;LMS addresses of all display lines.
             8630 ;This scrolls the characters.
             8640 ;
7959 A009    8650 CHGDL   LDY  #$09
795B B1B0    8660 DLOOP   LDA  (DLSTPT),Y
795D 18      8670         CLC
795E 65B9    8680         ADC  OFFLO
7960 91B0    8690         STA  (DLSTPT),Y
7962 C8      8700         INY
7963 B1B0    8710         LDA  (DLSTPT),Y
7965 65BA    8720         ADC  OFFHI
7967 91B0    8730         STA  (DLSTPT),Y
7969 C8      8740         INY
796A C8      8750         INY
796B C027    8760         CPY  #$27
796D D0EC    8770         BNE  DLOOP
             8780 ;
796F AD0206  8790 ENDISR  LDA  YPOSH
7972 4A      8800         LSR  A
7973 AD0106  8810         LDA  YPOSL
7976 6A      8820         ROR  A
7977 4A      8830         LSR  A
7978 4A      8840         LSR  A
7979 4A      8850         LSR  A
797A C911    8860         CMP  #$11
797C B004    8870         BCS  X39
797E A9FF    8880         LDA  #$FF
7980 3010    8890         BMI  X40
7982 C91A    8900 X39     CMP  #$1A
7984 9004    8910         BCC  X41
7986 A902    8920         LDA  #$02
7988 1008    8930         BPL  X40
798A 85BB    8940 X41     STA  TEMP1
798C E8      8950         INX
798D A91D    8960         LDA  #$1D
798F 38      8970         SEC
7990 E5BB    8980         SBC  TEMP1
7992 85BC    8990 X40     STA  CNT1
7994 A900    9000         LDA  #$00
7996 85BD    9010         STA  CNT2
7998 4C62E4  9020         JMP  XITVBV     exit vertical blank routine
             9030 ;
799B         9040         *=   $799C
799C 00      9050 JSTP    .BYTE 0,0,0,0,3,3,3,3
799D 00
799E 00
799F 00
79A0 03
79A1 03
79A2 03
79A3 03
79A4 02      9060         .BYTE 2,2,2,2,1,1,1,0
79A5 02
79A6 02
79A7 02
79A8 01
79A9 01
79AA 01
79AB 00
79AC 00      9070         .BYTE 0,0,3,3,2,2,1,0
79AD 00
79AE 03
79AF 03
79B0 02
79B1 02
79B2 01
79B3 00
79B4 02      9080 DEFNC   .BYTE 2,3,3,2,2,2,1,1,2,0
79B5 03
79B6 03
79B7 02
79B8 02
79B9 02
79BA 01
```

```
79BB 01
79BC 02
79BD 00
79BE          9090          *=   $79C0
              9100 ;
              9110 ;SUBROUTINE DWORDS
              9120 ;displays a single word from a long table of words
              9130 ;
79C0 0A       9140 DWORDS ASL  A
79C1 0A       9150        ASL  A
79C2 0A       9160        ASL  A
79C3 9015     9170        BCC  ENTRY2
79C5 AA       9180        TAX
79C6 BDBA58   9190 BOOP20 LDA  WORDS+256,X
79C9 38       9200        SEC
79CA E920     9210        SBC  #$20
79CC F00A     9220        BEQ  BNDW
79CE 995064   9230        STA  TXTWDW,Y
79D1 C8       9240        INY
79D2 E8       9250        INX
79D3 8A       9260        TXA
79D4 2907     9270        AND  #$07
79D6 D0EE     9280        BNE  BOOP20
79D8 C8       9290 BNDW   INY
79D9 60       9300        RTS
79DA AA       9310 ENTRY2 TAX             this is another entry point
79DB BDBA57   9320 LOOP20 LDA  WORDS,X
79DE 38       9330        SEC
79DF E920     9340        SBC  #$20
79E1 F00A     9350        BEQ  NDW
79E3 995064   9360        STA  TXTWDW,Y
79E6 C8       9370        INY
79E7 E8       9380        INX
79E8 8A       9390        TXA
79E9 2907     9400        AND  #$07
79EB D0EE     9410        BNE  LOOP20
79ED C8       9420 NDW    INY
79EE 60       9430        RTS
              9440 ;
              9450 ;
              9460 ;SUBROUTINE SWITCH FOR SWAPPING CORPS WITH TERRAIN
              9470 ;
79EF A900     9480 SWITCH LDA  #$00
79F1 85B3     9490        STA  MAPHI
79F3 A927     9500        LDA  #$27
79F5 38       9510        SEC
79F6 E5BF     9520        SBC  CHUNKY
79F8 0A       9530        ASL  A
79F9 26B3     9540        ROL  MAPHI
79FB 0A       9550        ASL  A
79FC 26B3     9560        ROL  MAPHI
79FE 0A       9570        ASL  A
79FF 26B3     9580        ROL  MAPHI
7A01 0A       9590        ASL  A
7A02 26B3     9600        ROL  MAPHI
7A04 8D1406   9610        STA  TEMPLO
7A07 A6B3     9620        LDX  MAPHI
7A09 8E1506   9630        STX  TEMPHI
7A0C 0A       9640        ASL  A
7A0D 26B3     9650        ROL  MAPHI
7A0F 18       9660        CLC
7A10 6D1406   9670        ADC  TEMPLO
7A13 85B2     9680        STA  MAPLO
7A15 A5B3     9690        LDA  MAPHI
7A17 6D1506   9700        ADC  TEMPHI
7A1A 6965     9710        ADC  #$65
7A1C 85B3     9720        STA  MAPHI
7A1E A92E     9730        LDA  #46
7A20 38       9740        SEC
7A21 E5BE     9750        SBC  CHUNKX
7A23 A8       9760        TAY
7A24 B1B2     9770        LDA  (MAPLO),Y
7A26 A6B4     9780        LDX  CORPS
7A28 F00A     9790        BEQ  X34
7A2A 48       9800        PHA
7A2B BD7C56   9810        LDA  SWAP,X
7A2E 91B2     9820        STA  (MAPLO),Y
7A30 68       9830        PLA
7A31 9D7C56   9840        STA  SWAP,X
7A34 60       9850 X34    RTS
              9860 ;
              9870 ;SUBROUTINE CLRP1
              9880 ;clears the arrow player
              9890 ;
7A35 A900     9900 CLRP1  LDA  #$00
7A37 AC1906   9910        LDY  STEPY
7A3A 88       9920        DEY
7A3B AA       9930        TAX
7A3C C080     9940 LOOP23 CPY  #$80
7A3E B003     9950        BCS  X22
7A40 9980D2   9960        STA  PLYR1,Y
7A43 C8       9970 X22    INY
7A44 E8       9980        INX
7A45 E00B     9990        CPX  #$0B
7A47 D0F3     010000      BNE  LOOP23
7A49 60       010010      RTS
              010020 ;
              010030 ;SUBROUTINE CLRP2
              010040 ;clears the maltakreuze
              010050 ;
7A4A A900     010060 CLRP2 LDA #$00
7A4C AC2106   010070       LDY  KRZY
7A4F AA       010080       TAX
7A50 C080     010090 LOOP25 CPY #$80
```

```
7A52 B003   010100       BCS  X42
7A54 990053 010110       STA  PLYR2,Y
7A57 C8     010120 X42   INY
7A58 E8     010130       INX
7A59 E00A   010140       CPX  #$0A
7A5B D0F3   010150       BNE  LOOP25
7A5D 60     010160       RTS
            010170 ;
            010180 ;SUBROUTINE ERRCLR
            010190 ;clears sound and the text window
            010200 ;
7A5E AD2406 010210 ERRCLR LDA ERRFLG
7A61 1010   010220       BPL  ENDERR
7A63 A900   010230       LDA  #$00
7A65 8D2406 010240       STA  ERRFLG
7A68 A086   010250       LDY  #$86
7A6A A21F   010260       LDX  #$1F
7A6C 995064 010270 LOOP29 STA TXTWDW,Y
7A6F 88     010280       DEY
7A70 CA     010290       DEX
7A71 10F9   010300       BPL  LOOP29
7A73 60     010310 ENDERR RTS
            010320 ;
7A74 C0     010330 BITTAB .BYTE $C0,3,$C,$30
7A75 03
7A76 0C
7A77 30
7A78 04     010340 ROTARR .BYTE 4,9,14,19,24
7A79 09
7A7A 0E
7A7B 13
7A7C 18
7A7D 03     010350       .BYTE 3,8,13,18,23
7A7E 08
7A7F 0D
7A80 12
7A81 17
7A82 02     010360       .BYTE 2,7,12,17,22
7A83 07
7A84 0C
7A85 11
7A86 16
7A87 01     010370       .BYTE 1,6,11,16,21
7A88 06
7A89 0B
7A8A 10
7A8B 15
7A8C 00     010380       .BYTE 0,5,10,15,20
7A8D 05
7A8E 0A
7A8F 0F
7A90 14
```

```
7A91        010390 OBJX *=   *+104
            010400 ;
            010410 ;From here to $7B00 is expansion RAM
            010420 ;
            010430 ;This is the DLI routine
            010440 ;
7AF9        010450       *=   $7B00
7B00 48     010460 DLISRV PHA
7B01 8A     010470       TXA
7B02 48     010480       PHA
7B03 E6BD   010490       INC  CNT2
7B05 A5BD   010500       LDA  CNT2
7B07 C5BC   010510       CMP  CNT1
7B09 D014   010520       BNE  OVER1
7B0B A262   010530       LDX  #$62       map DLI
7B0D A928   010540       LDA  #$28
7B0F 454F   010550       EOR  COLRSH
7B11 254E   010560       AND  DRKMSK
7B13 8D0AD4 010570       STA  WSYNC
7B16 8E09D4 010580       STX  CHBASE
7B19 8D16D0 010590       STA  COLPF0
7B1C 4CAE7B 010600       JMP  DLIOUT
            010610 ;
7B1F C90F   010620 OVER1 CMP #$0F
7B21 D019   010630       BNE  OVER6
7B23 A93A   010640       LDA  #$3A
7B25 454F   010650       EOR  COLRSH
7B27 254E   010660       AND  DRKMSK
7B29 AA     010670       TAX
7B2A A900   010680       LDA  #$00
7B2C 454F   010690       EOR  COLRSH
7B2E 254E   010700       AND  DRKMSK
7B30 8D0AD4 010710       STA  WSYNC
7B33 8E18D0 010720       STX  COLPF2
7B36 8D17D0 010730       STA  COLPF1
7B39 4CAE7B 010740       JMP  DLIOUT
            010750 ;
7B3C C901   010760 OVER6 CMP #$01
7B3E D01F   010770       BNE  OVER2
7B40 AD0506 010780       LDA  TRCOLR     green tree color
7B43 454F   010790       EOR  COLRSH
7B45 254E   010800       AND  DRKMSK
7B47 AA     010810       TAX
7B48 A91A   010820       LDA  #$1A       yellow band at top of map
7B4A 454F   010830       EOR  COLRSH
7B4C 254E   010840       AND  DRKMSK
7B4E 8D0AD4 010850       STA  WSYNC
7B51 8D1AD0 010860       STA  COLBAK
7B54 8E16D0 010870       STX  COLPF0
7B57 A960   010880       LDA  #$60
7B59 8D09D4 010890       STA  CHBASE
7B5C 4CAE7B 010900       JMP  DLIOUT
```

```
                010910 ;
7B5F C903       010920 OVER2 CMP #$03
7B61 D010       010930       BNE  OVER3
7B63 AD0606     010940       LDA  EARTH      top of map
7B66 454F       010950       EOR  COLRSH
7B68 254E       010960       AND  DRKMSK
7B6A 8D0AD4     010970       STA  WSYNC
7B6D 8D1AD0     010980       STA  COLBAK
7B70 4CAE7B     010990       JMP  DLIOUT
                011000 ;
7B73 C90D       011010 OVER3 CMP #$0D
7B75 D014       011020       BNE  OVER4
7B77 A2E0       011030       LDX  #$E0       bottom of map
7B79 A922       011040       LDA  #$22
7B7B 454F       011050       EOR  COLRSH
7B7D 254E       011060       AND  DRKMSK
7B7F 8D0AD4     011070       STA  WSYNC
7B82 8D18D0     011080       STA  COLPF2
7B85 8E09D4     011090       STX  CHBASE
7B88 4CAE7B     011100       JMP  DLIOUT
                011110 ;
7B8B C90E       011120 OVER4 CMP #$0E
7B8D D00F       011130       BNE  OVER5
7B8F A98A       011140       LDA  #$8A       bright blue strip
7B91 454F       011150       EOR  COLRSH
7B93 254E       011160       AND  DRKMSK
7B95 8D0AD4     011170       STA  WSYNC
7B98 8D1AD0     011180       STA  COLBAK
7B9B 4CAE7B     011190       JMP  DLIOUT
                011200 ;
7B9E C910       011210 OVER5 CMP #$10
7BA0 D00C       011220       BNE  DLIOUT
7BA2 A9D4       011230       LDA  #$D4       green bottom
7BA4 454F       011240       EOR  COLRSH
7BA6 254E       011250       AND  DRKMSK
7BA8 48         011260       PHA             some extra delay
7BA9 68         011270       PLA
7BAA EA         011280       NOP
7BAB 8D1AD0     011290       STA  COLBAK
                011300 ;
7BAE 68         011310 DLIOUT PLA
7BAF AA         011320       TAX
7BB0 68         011330       PLA
7BB1 40         011340       RTI
                011350 ;
                011360 ;SUBROUTINE DNUMBR
                011370 ;displays a number with leading zero suppress
                011380 ;
7BB2 AA         011390 DNUMBR TAX
7BB3 18         011400       CLC
7BB4 BD085A     011410       LDA  HDIGIT,X
7BB7 F007       011420       BEQ  X36
7BB9 6910       011430       ADC  #$10
7A95064 011440       STA  TXTWDW,Y
7BBE C8         011450       INY
7BBF 38         011460       SEC
7BC0 BD085B     011470 X36   LDA  TDIGIT,X
7BC3 B002       011480       BCS  X38
7BC5 F007       011490       BEQ  X37
7BC7 18         011500 X38   CLC
7BC8 6910       011510       ADC  #$10
7BCA 995064     011520       STA  TXTWDW,Y
7BCD C8         011530       INY
7BCE BD085C     011540 X37   LDA  ODIGIT,X
7BD1 18         011550       CLC
7BD2 6910       011560       ADC  #$10
7BD4 995064     011570       STA  TXTWDW,Y
7BD7 C8         011580       INY
7BD8 60         011590       RTS
                011600 ;
7BD9 00         011610 NDX   .BYTE 0,1,2,3,4,9,14,19
7BDA 01
7BDB 02
7BDC 03
7BDD 04
7BDE 09
7BDF 0E
7BE0 13
7BE1 18         011620       .BYTE 24,23,22,21,20,15,10,5
7BE2 17
7BE3 16
7BE4 15
7BE5 14
7BE6 0F
7BE7 0A
7BE8 05
7BE9 06         011630       .BYTE 6,7,8,13,18,17,16,11
7BEA 07
7BEB 08
7BEC 0D
7BED 12
7BEE 11
7BEF 10
7BF0 0B
7BF1 01         011640 YINC .BYTE 1
7BF2 00         011650 XINC .BYTE 0,$FF,0,1
7BF3 FF
7BF4 00
7BF5 01
7BF6 01         011660 OFFNC .BYTE 1,1,1,1,1,1,2,2,1,0
7BF7 01
7BF8 01
7BF9 01
7BFA 01
```

```
7BFB 01
7BFC 02
7BFD 02
7BFE 01
7BFF 00
7C00          011670        .END
```

```
                10 ;EFT VERSION 1.8M (MAINLINE) 11/30/81 COPYRIGHT CHRIS CRAWFORD 1981
                20 ;
                30 ;Page zero RAM
00B0            40 DLSTPT   =    $B0        Zero page pointer to display list
00B4            50 CORPS    =    $B4
0000            60          *=   $BE
00BE            70 CHUNKX   *=   *+1        cursor coordinates (pixel frame)
00BF            80 CHUNKY   *=   *+1
                90 ;
                0100 ;These locations are for the mainline routines
                0110 ;
00C0            0120 MAPPTR *=   *+2
00C2            0130 ARMY   *=   *+1
00C3            0140 UNITNO *=   *+1
00C4            0150 DEFNDR *=   *+1
00C5            0160 TEMPR  *=   *+1
00C6            0170 TEMPZ  *=   *+1
00C7            0180 ACCLO  *=   *+1
00C8            0190 ACCHI  *=   *+1
00C9            0200 TURN   *=   *+1
00CA            0210 LAT    *=   *+1
00CB            0220 LONG   *=   *+1
00CC            0230 RFR    *=   *+1
00CD            0240 TRNTYP *=   *+1
00CE            0250 SQVAL  *=   *+1
                0260 ;
                0270 ;OS locations (see OS manual)
                0280 ;
022F            0290 SDMCTL =    $022F
0230            0300 DLSTLO =    $0230      Existing OS pointer to display list
0231            0310 DLSTHI =    $0231
026F            0320 GPRIOR =    $026F
02C0            0330 PCOLR0 =    $02C0
                0340 ;
                0350 ;HARDWARE LOCATIONS
                0360 ;
D000            0370 HPOSP0 =    $D000
D008            0380 SIZEP0 =    $D008
D01A            0390 COLBAK =    $D01A
D01D            0400 GRACTL =    $D01D
D20A            0410 RANDOM =    $D20A
D404            0420 HSCROL =    $D404
D405            0430 VSCROL =    $D405
D407            0440 PMBASE =    $D407
D40E            0450 NMIEN  =    $D40E
E45C            0460 SETVBV =    $E45C
                0470 ;
                0480 ;Page 6 usage
                0490 ;
00CF            0500         *=  $0600
                0510 ;first come locations used by the interrupt service routine
0600            0520 XPOSL  *=   *+5        Horizontal position of screen window
0605            0530 TRCOLR *=   *+1
0606            0540 EARTH  *=   *+1
0607            0550 ICELAT *=   *+1
0608            0560 SEASN1 *=   *+1
0609            0570 SEASN2 *=   *+1
060A            0580 SEASN3 *=   *+1
060B            0590 DAY    *=   *+1
060C            0600 MONTH  *=   *+1
060D            0610 YEAR   *=   *+1
060E            0620 BUTFLG *=   *+1
060F            0630 BUTMSK *=   *+1
                0640 ;
                0650 ;THESE VALUES ARE USED BY MAINLINE ROUTINE ONLY
                0660 ;
0610            0670         *=  $62A
062A            0680 OLDLAT *=   *+1
062B            0690 TRNCOD *=   *+1
062C            0700 TLO    *=   *+1
062D            0710 THI    *=   *+1
062E            0720 TICK   *=   *+1
062F            0730 UNTCOD *=   *+1
0630            0740 UNTCD1 *=   *+1
                0750 ;
068F            0760 HANDCP =    $68F
0694            0770 ZOC    =    $694
0697            0780 VICTRY =    $697
                0790 ;
                0800 ;declarations of routines in other modules
                0810 ;
5140            0820 CHKZOC =    $5140
5091            0830 LOGSTC =    $5091
7BB2            0840 DNUMBR =    $7BB2
79C0            0850 DWORDS =    $79C0
79EF            0860 SWITCH =    $79EF
7BF1            0870 YINC   =    $7BF1
7BF2            0880 XINC   =    $7BF2
                0890 ;
0631            0900         *=  $5200
5200            0910 PLYR0  *=   *+512
5400            0920 CORPSX *=   *+159      x-coords of all units (pixel frame)
549F            0930 CORPSY *=   *+159      y-coords of all units (pixel frame)
553E            0940 MSTRNG *=   *+159      muster strengths
55DD            0950 CSTRNG *=   *+159      combat strengths
567C            0960 SWAP   *=   *+159      terrain code underneath unit
571B            0970 ARRIVE *=   *+159      turn of arrival
57BA            0980         *=  $5C08
5C08            0990 ODIGIT *=   *+256
5D08            1000 TXTTBL *=   *+96       more text
5D68            1010 MONLEN *=   *+13       table of month lengths
5D75            1020 HMORDS *=   *+159      how many orders each unit has in queue
5E14            1030 WHORDS *=   *+159      what the orders are
```

```
5EB3          1040 WHORDH *=   *+159
5F52          1050        *=   $5FE2
5FE2          1060 XADD   *=   *+4        offsets for moving arrow
5FE6          1070 YADD   *=   *+4
5FEA          1080 TRTAB  *=   *+13
5FF7          1090 MLTKRZ *=   *+8        maltese cross shape
              1100 ;
5FFF          1110        *=   $6450
6450          1120 TXTWDW *=   $6CB1
6CB1          1130 STKTAB *=   *+16       a joystick decoding table
6CC1          1140 SSNCOD *=   *+12
6CCD          1150 TRNTAB *=   *+60
6D09          1160 BHX1   *=   *+22
6D1F          1170 BHY1   *=   *+22
6D35          1180 BHX2   *=   *+22
6D4B          1190 BHY2   *=   *+22
6D61          1200 EXEC   *=   *+159
              1210 ;
              1220 ;This is the initialization program
              1230 ;The program begins here
              1240 ;
6E00          1250        *=   $6E00
              1260 ;
6E00 A208     1270        LDX  #$08
6E02 BDB673   1280 BOOP99 LDA  ZPVAL,X    initialize page zero values
6E05 95B0     1290        STA  DLSTPT,X
6E07 BDCF73   1300        LDA  COLTAB,X
6E0A 9DC002   1310        STA  PCOLR0,X
6E0D CA       1320        DEX
6E0E 10F2     1330        BPL  BOOP99
              1340 ;
6E10 A20F     1350        LDX  #$0F
6E12 BDBF73   1360 BOOP98 LDA  PSXVAL,X   initialize page six values
6E15 9D0006   1370        STA  XPOSL,X
6E18 CA       1380        DEX
6E19 10F7     1390        BPL  BOOP98
              1400 ;
6E1B A900     1410        LDA  #$00
6E1D 8D3002   1420        STA  DLSTLO
6E20 8D04D4   1430        STA  HSCROL
6E23 8D05D4   1440        STA  VSCROL
6E26 A5B1     1450        LDA  DLSTPT+1
6E28 8D3102   1460        STA  DLSTHI
              1470 ;
6E2B A200     1480        LDX  #$00
6E2D BD3E55   1490 LOOP22 LDA  MSTRNG,X
6E30 9DDD55   1500        STA  CSTRNG,X
6E33 A900     1510        LDA  #$00
6E35 9D755D   1520        STA  HMORDS,X
6E38 A9FF     1530        LDA  #$FF
6E3A 9D616D   1540        STA  EXEC,X
6E3D E8       1550        INX
6E3E E0A0     1560        CPX  #$A0
6E40 D0EB     1570        BNE  LOOP22
              1580 ;
              1590 ;
              1600 ;Now set up player window
              1610 ;
6E42 A950     1620        LDA  #$50
6E44 8D07D4   1630        STA  PMBASE
              1640 ;
              1650 ;here follow various initializations
              1660 ;
6E47 A92F     1670        LDA  #$2F
6E49 8D2F02   1680        STA  SDMCTL
6E4C A903     1690        LDA  #$03
6E4E 8D1DD0   1700        STA  GRACTL
6E51 A978     1710        LDA  #$78
6E53 8D00D0   1720        STA  HPOSP0
6E56 A901     1730        LDA  #$01
6E58 8D8F06   1740        STA  HANDCP
6E5B 8D6F02   1750        STA  GPRIOR
6E5E 8D08D0   1760        STA  SIZEP0
6E61 A233     1770        LDX  #$33
              1780 ;
6E63 A9FF     1790        LDA  #$FF
6E65 9D0052   1800        STA  PLYR0,X
6E68 E8       1810        INX
6E69 9D0052   1820        STA  PLYR0,X
6E6C E8       1830        INX
6E6D A981     1840        LDA  #$81
6E6F 9D0052   1850 LOOP2  STA  PLYR0,X
6E72 E8       1860        INX
6E73 E03F     1870        CPX  #$3F
6E75 D0F8     1880        BNE  LOOP2
6E77 A9FF     1890        LDA  #$FF
6E79 9D0052   1900        STA  PLYR0,X
6E7C 85C9     1910        STA  TURN
6E7E E8       1920        INX
6E7F 9D0052   1930        STA  PLYR0,X
              1940 ;
              1950 ;Now enable deferred vertical blank interrupt
              1960 ;
6E82 A000     1970        LDY  #$00
6E84 A274     1980        LDX  #$74
6E86 A907     1990        LDA  #$07
6E88 205CE4   2000        JSR  SETVBV
6E8B A900     2010        LDA  #$00       This is DLI vector (low byte)
6E8D 8D0002   2020        STA  $0200
6E90 A97B     2030        LDA  #$7B
6E92 8D0102   2040        STA  $0201
6E95 A9C0     2050        LDA  #$C0
6E97 8D0ED4   2060        STA  NMIEN      Turn interrupts on
              2070 ;
```

```
6E9A E6C9   2080 NEWTRN INC  TURN
            2090 ;
            2100 ;first do calendar calculations
            2110 ;
6E9C AD0B06 2120        LDA  DAY
6E9F 18     2130        CLC
6EA0 6907   2140        ADC  #07
6EA2 AE0C06 2150        LDX  MONTH
6EA5 DD685D 2160        CMP  MONLEN,X
6EA8 F027   2170        BEQ  X28
6EAA 9025   2180        BCC  X28
6EAC E002   2190        CPX  #$02
6EAE D00A   2200        BNE  X96
6EB0 AC0D06 2210        LDY  YEAR
6EB3 C02C   2220        CPY  #44
6EB5 D003   2230        BNE  X96
6EB7 38     2240        SEC
6EB8 E901   2250        SBC  #$01
6EBA 38     2260 X96    SEC
6EBB FD685D 2270        SBC  MONLEN,X
6EBE E8     2280        INX
6EBF E00D   2290        CPX  #13
6EC1 D005   2300        BNE  X29
6EC3 EE0D06 2310        INC  YEAR
6EC6 A201   2320        LDX  #01
6EC8 8E0C06 2330 X29    STX  MONTH
6ECB BCEA5F 2340        LDY  TRTAB,X
6ECE 8C0506 2350        STY  TRCOLR
6ED1 8D0B06 2360 X28    STA  DAY
6ED4 A093   2370        LDY  #$93
6ED6 A900   2380        LDA  #$00
6ED8 995064 2390 LOOP13 STA  TXTWDW,Y
6EDB C8     2400        INY
6EDC C0A7   2410        CPY  #$A7
6EDE D0F8   2420        BNE  LOOP13
6EE0 A093   2430        LDY  #$93
6EE2 8A     2440        TXA
6EE3 18     2450        CLC
6EE4 6910   2460        ADC  #$10
6EE6 20C079 2470        JSR  DWORDS
6EE9 AD0B06 2480        LDA  DAY
6EEC 20B27B 2490        JSR  DNUMBR
6EEF A90C   2500        LDA  #$0C
6EF1 995064 2510        STA  TXTWDW,Y
6EF4 C8     2520        INY
6EF5 C8     2530        INY
6EF6 A911   2540        LDA  #$11
6EF8 995064 2550        STA  TXTWDW,Y
6EFB C8     2560        INY
6EFC A919   2570        LDA  #$19
6EFE 995064 2580        STA  TXTWDW,Y
6F01 C8     2590        INY
6F02 AE0D06 2600        LDX  YEAR
6F05 A914   2610        LDA  #$14
6F07 995064 2620        STA  TXTWDW,Y
6F0A C8     2630        INY
6F0B BD085C 2640        LDA  ODIGIT,X
6F0E 18     2650        CLC
6F0F 6910   2660        ADC  #$10
6F11 995064 2670        STA  TXTWDW,Y
            2680 ;
            2690 ;now do season calculations
            2700 ;
6F14 AD0C06 2710        LDA  MONTH
6F17 C904   2720        CMP  #$04
6F19 D017   2730        BNE  X87
6F1B A902   2740        LDA  #$02
6F1D 8D0606 2750        STA  EARTH
6F20 A940   2760        LDA  #$40
6F22 8D0806 2770        STA  SEASN1
6F25 A901   2780        LDA  #$01
6F27 8D0A06 2790        STA  SEASN3
6F2A A900   2800        LDA  #$00
6F2C 8D0906 2810        STA  SEASN2
6F2F 4CEB6F 2820        JMP  ENDSSN
6F32 C90A   2830 X87    CMP  #$0A
6F34 D008   2840        BNE  X88
6F36 A902   2850        LDA  #$02
6F38 8D0606 2860        STA  EARTH
6F3B 4CEB6F 2870        JMP  ENDSSN
6F3E C905   2880 X88    CMP  #$05
6F40 D008   2890        BNE  X89
6F42 A910   2900        LDA  #$10
6F44 8D0606 2910        STA  EARTH
6F47 4CEB6F 2920        JMP  ENDSSN
6F4A C90B   2930 X89    CMP  #$0B
6F4C D008   2940        BNE  X90
6F4E A90A   2950        LDA  #$0A
6F50 8D0606 2960        STA  EARTH
6F53 4C716F 2970        JMP  X91
6F56 C901   2980 X90    CMP  #$01
6F58 D010   2990        BNE  X92
6F5A A980   3000        LDA  #$80
6F5C 8D0806 3010        STA  SEASN1
6F5F A9FF   3020        LDA  #$FF
6F61 8D0906 3030        STA  SEASN2
6F64 8D0A06 3040        STA  SEASN3
6F67 4CEB6F 3050        JMP  ENDSSN
6F6A C903   3060 X92    CMP  #$03
6F6C F003   3070        BEQ  X91
6F6E 4CEB6F 3080        JMP  ENDSSN
            3090 ;
            3100 ;
            3110 ;freeze those rivers, baby
```

```
                3120 ;
6F71 AD0AD2     3130 X91    LDA  RANDOM
6F74 2907       3140        AND  #$07
6F76 18         3150        CLC
6F77 6907       3160        ADC  #$07
6F79 4D0906     3170        EOR  SEASN2
6F7C 85C5       3180        STA  TEMPR
6F7E AD0706     3190        LDA  ICELAT
6F81 8D2A06     3200        STA  OLDLAT
6F84 38         3210        SEC
6F85 E5C5       3220        SBC  TEMPR
6F87 F002       3230        BEQ  X95
6F89 1002       3240        BPL  X94
6F8B A901       3250 X95    LDA  #$01
6F8D C927       3260 X94    CMP  #$27
6F8F 9002       3270        BCC  X93
6F91 A927       3280        LDA  #$27
6F93 8D0706     3290 X93    STA  ICELAT
6F96 A901       3300        LDA  #$01
6F98 85BE       3310        STA  CHUNKX
6F9A 85CB       3320        STA  LONG
6F9C AD2A06     3330        LDA  OLDLAT
6F9F 85BF       3340        STA  CHUNKY
6FA1 85CA       3350        STA  LAT
                3360 ;
6FA3 204072     3370 LOOP40 JSR  TERR
                3380 ;
6FA6 293F       3390        AND  #$3F
6FA8 C90B       3400        CMP  #$0B
6FAA 901D       3410        BCC  NOTCH
6FAC C929       3420        CMP  #$29
6FAE B019       3430        BCS  NOTCH
6FB0 A6BF       3440        LDX  CHUNKY
6FB2 E00E       3450        CPX  #$0E
6FB4 B004       3460        BCS  DOTCH
6FB6 C923       3470        CMP  #$23
6FB8 B00F       3480        BCS  NOTCH
6FBA 0D0806     3490 DOTCH  ORA  SEASN1
6FBD A6C3       3500        LDX  UNITNO
6FBF F006       3510        BEQ  X86
6FC1 9D7C56     3520        STA  SWAP,X
6FC4 4CC96F     3530        JMP  NOTCH
6FC7 91C0       3540 X86    STA  (MAPPTR),Y
6FC9 E6BE       3550 NOTCH  INC  CHUNKX
6FCB A5BE       3560        LDA  CHUNKX
6FCD 85CB       3570        STA  LONG
6FCF C92E       3580        CMP  #46
6FD1 D0D0       3590        BNE  LOOP40
6FD3 A900       3600        LDA  #$00
6FD5 85BE       3610        STA  CHUNKX
6FD7 85CB       3620        STA  LONG
6FD9 A5BF       3630        LDA  CHUNKY
6FDB CD0706     3640        CMP  ICELAT
6FDE F00B       3650        BEQ  ENDSSN
6FE0 38         3660        SEC
6FE1 ED0A06     3670        SBC  SEASN3
6FE4 85BF       3680        STA  CHUNKY
6FE6 85CA       3690        STA  LAT
6FE8 4CA36F     3700        JMP  LOOP40
                3710 ;
6FEB A29E       3720 ENDSSN LDX  #$9E       any reinforcements?
6FED BD1B57     3730 LOOP14 LDA  ARRIVE,X
6FF0 C5C9       3740        CMP  TURN
6FF2 D02C       3750        BNE  X33
6FF4 BD0054     3760        LDA  CORPSX,X
6FF7 85BE       3770        STA  CHUNKX
6FF9 85CB       3780        STA  LONG
6FFB BD9F54     3790        LDA  CORPSY,X
6FFE 85BF       3800        STA  CHUNKY
7000 85CA       3810        STA  LAT
7002 86B4       3820        STX  CORPS
7004 204672     3830        JSR  TERRB
7007 F00F       3840        BEQ  SORRY
7009 E037       3850        CPX  #$37
700B B005       3860        BCS  A51
700D A90A       3870        LDA  #$0A
700F 8D7464     3880        STA  TXTWDW+36
7012 20EF79     3890 A51    JSR  SWITCH
7015 4C2070     3900        JMP  X33
7018 A5C9       3910 SORRY  LDA  TURN
701A 18         3920        CLC
701B 6901       3930        ADC  #$01
701D 9D1B57     3940        STA  ARRIVE,X
7020 CA         3950 X33    DEX
7021 D0CA       3960        BNE  LOOP14
                3970 ;
7023 A29E       3980 X31    LDX  #$9E
7025 86C2       3990 LOOPF  STX  ARMY
7027 209150     4000        JSR  LOGSTC     logistics subroutine
702A A6C2       4010        LDX  ARMY
702C CA         4020        DEX
702D D0F6       4030        BNE  LOOPF   K>         4040 ;
                4050 ;calculate some points
                4060 ;
702F A900       4070        LDA  #$00
7031 85C7       4080        STA  ACCLO
7033 85C8       4090        STA  ACCHI
7035 A201       4100        LDX  #$01
7037 A930       4110 LOOPB  LDA  #$30
7039 38         4120        SEC
703A FD0054     4130        SBC  CORPSX,X
703D 85C5       4140        STA  TEMPR
703F BD3E55     4150        LDA  MSTRNG,X
```

```
7042 4A      4160          LSR  A
7043 F012    4170          BEQ  A01
7045 A8      4180          TAY
7046 A900    4190          LDA  #$00
7048 18      4200          CLC
7049 65C5    4210 LOOPA    ADC  TEMPR
704B 9007    4220          BCC  A0
704D E6C8    4230          INC  ACCHI
704F 18      4240          CLC
7050 D002    4250          BNE  A0
7052 C6C8    4260          DEC  ACCHI
7054 88      4270 A0       DEY
7055 D0F2    4280          BNE  LOOPA
7057 E8      4290 A01      INX
7058 E037    4300          CPX  #$37
705A D0DB    4310          BNE  LOOPB
             4320 ;
705C BD0054  4330 LOOPC    LDA  CORPSX,X
705F 85C5    4340          STA  TEMPR
7061 BDDD55  4350          LDA  CSTRNG,X
7064 4A      4360          LSR  A
7065 4A      4370          LSR  A
7066 4A      4380          LSR  A
7067 F012    4390          BEQ  A02
7069 A8      4400          TAY
706A A900    4410          LDA  #$00
706C 18      4420          CLC
706D 65C5    4430 LOOPD    ADC  TEMPR
706F 9007    4440          BCC  A03
7071 E6C7    4450          INC  ACCLO
7073 18      4460          CLC
7074 D002    4470          BNE  A03
7076 C6C7    4480          DEC  ACCLO
7078 88      4490 A03      DEY
7079 D0F2    4500          BNE  LOOPD
707B E8      4510 A02      INX
707C E09E    4520          CPX  #$9E
707E D0DC    4530          BNE  LOOPC
             4540 ;
7080 A5C8    4550          LDA  ACCHI
7082 38      4560          SEC
7083 E5C7    4570          SBC  ACCLO
7085 B002    4580          BCS  A04
7087 A900    4590          LDA  #$00
7089 A203    4600 A04      LDX  #$03
708B BCEA71  4610 LOOPG    LDY  MOSCOW,X
708E F008    4620          BEQ  A15
7090 18      4630          CLC
7091 7DD873  4640          ADC  MPTS,X
7094 9002    4650          BCC  A15
7096 A9FF    4660          LDA  #$FF
7098 CA      4670 A15      DEX
```

```
7099 10F0    4680          BPL  LOOPG
             4690 ;
709B AE8F06  4700          LDX  HANDCP     was handicap option used?
709E D001    4710          BNE  A23        no
70A0 4A      4720          LSR  A          yes, halve score
70A1 A005    4730 A23      LDY  #$05
70A3 20B27B  4740          JSR  DNUMBR
70A6 A900    4750          LDA  #$00
70A8 995064  4760          STA  TXTWDW,Y
70AB A5C9    4770          LDA  TURN
70AD C928    4780          CMP  #$28
70AF D008    4790          BNE  Z00
70B1 A901    4800          LDA  #$01       end of game
70B3 20E473  4810          JSR  TXTMSG
70B6 4CB670  4820 FINI     JMP  FINI       hang up
             4830 ;
             4840 ;
70B9 A900    4850 Z00      LDA  #$00
70BB 8D0F06  4860          STA  BUTMSK
70BE 85B4    4870          STA  CORPS
70C0 20E473  4880          JSR  TXTMSG
70C3 200047  4890          JSR  $4700      artificial intelligence routine
70C6 A901    4900          LDA  #$01
70C8 8D0F06  4910          STA  BUTMSK
70CB A902    4920          LDA  #$02
70CD 20E473  4930          JSR  TXTMSG
             4940 ;
             4950 ;movement execution phase
             4960 ;
70D0 A900    4970          LDA  #$00
70D2 8D2E06  4980          STA  TICK
70D5 A29E    4990          LDX  #$9E
70D7 86C2    5000 LOOP31   STX  ARMY
70D9 20D172  5010          JSR  DINGO      determine first execution time
70DC CA      5020          DEX
70DD D0F8    5030          BNE  LOOP31
             5040 ;
70DF A29E    5050 LOOP33   LDX  #$9E
70E1 86C2    5060 LOOP32   STX  ARMY
70E3 BD3E55  5070          LDA  MSTRNG,X
70E6 38      5080          SEC
70E7 FDDD55  5090          SBC  CSTRNG,X
70EA C902    5100          CMP  #$02
70EC 900B    5110          BCC  Y30
70EE FEDD55  5120          INC  CSTRNG,X
70F1 CD0AD2  5130          CMP  RANDOM
70F4 9003    5140          BCC  Y30
70F6 FEDD55  5150          INC  CSTRNG,X
70F9 BD616D  5160 Y30      LDA  EXEC,X
70FC 3045    5170          BMI  A60
70FE CD2E06  5180          CMP  TICK
7101 D040    5190          BNE  A60
```

```
7103 BD145E 5200         LDA  WHORDS,X
7106 2903   5210         AND  #$03
7108 A8     5220         TAY
7109 BD0054 5230         LDA  CORPSX,X
710C 18     5240         CLC
710D 79F27B 5250         ADC  XINC,Y
7110 85CB   5260         STA  LONG
7112 85C7   5270         STA  ACCLO
7114 BD9F54 5280         LDA  CORPSY,X
7117 18     5290         CLC
7118 79F17B 5300         ADC  YINC,Y
711B 85CA   5310         STA  LAT
711D 85C8   5320         STA  ACCHI
711F 204072 5330         JSR  TERR
7122 A5C3   5340         LDA  UNITNO
7124 F02A   5350         BEQ  DOMOVE
7126 C937   5360         CMP  #$37
7128 9008   5370         BCC  GERMAN
712A A5C2   5380         LDA  ARMY
712C C937   5390         CMP  #$37
712E B008   5400         BCS  TRJAM
7130 9014   5410         BCC  COMBAT
7132 A5C2   5420 GERMAN  LDA  ARMY
7134 C937   5430         CMP  #$37
7136 B00E   5440         BCS  COMBAT
7138 A6C2   5450 TRJAM   LDX  ARMY
713A AD2E06 5460         LDA  TICK
713D 18     5470         CLC
713E 6902   5480         ADC  #$02
7140 9D616D 5490         STA  EXEC,X
7143 4CD271 5500 A60     JMP  Y06
7146 20D84E 5510 COMBAT  JSR  $4ED8
7149 AD9706 5520         LDA  VICTRY
714C F0F5   5530         BEQ  A60
714E D02E   5540         BNE  Z94
7150 A6C2   5550 DOMOVE  LDX  ARMY
7152 86B4   5560         STX  CORPS
7154 BD9F54 5570         LDA  CORPSY,X
7157 85BF   5580         STA  CHUNKY
7159 85CA   5590         STA  LAT
715B BD0054 5600         LDA  CORPSX,X
715E 85BE   5610         STA  CHUNKX
7160 85CB   5620         STA  LONG
7162 204051 5630         JSR  CHKZOC
7165 A5C8   5640         LDA  ACCHI
7167 85CA   5650         STA  LAT
7169 A5C7   5660         LDA  ACCLO
716B 85CB   5670         STA  LONG
716D AD9406 5680         LDA  ZOC
7170 C902   5690         CMP  #$02
7172 900A   5700         BCC  Z94
7174 204051 5710         JSR  CHKZOC
7177 AD9406 5720         LDA  ZOC
717A C902   5730         CMP  #$02
717C B0BA   5740         BCS  TRJAM
717E 20EF79 5750 Z94     JSR  SWITCH
7181 A6B4   5760         LDX  CORPS
7183 A5CA   5770         LDA  LAT
7185 85BF   5780         STA  CHUNKY
7187 9D9F54 5790         STA  CORPSY,X
718A A5CB   5800         LDA  LONG
718C 85BE   5810         STA  CHUNKX
718E 9D0054 5820         STA  CORPSX,X
7191 20EF79 5830         JSR  SWITCH
7194 A6C2   5840         LDX  ARMY
7196 A9FF   5850         LDA  #$FF
7198 9D616D 5860         STA  EXEC,X
719B DE755D 5870         DEC  HMORDS,X
719E F032   5880         BEQ  Y06
71A0 5EB35E 5890         LSR  WHORDH,X
71A3 7E145E 5900         ROR  WHORDS,X
71A6 5EB35E 5910         LSR  WHORDH,X
71A9 7E145E 5920         ROR  WHORDS,X
71AC A003   5930         LDY  #$03
71AE BD0054 5940 LOOPH   LDA  CORPSX,X
71B1 D9DC73 5950         CMP  MOSCX,Y
71B4 D013   5960         BNE  A18
71B6 BD9F54 5970         LDA  CORPSY,X
71B9 D9E073 5980         CMP  MOSCY,Y
71BC D00B   5990         BNE  A18
71BE A9FF   6000         LDA  #$FF
71C0 E037   6010         CPX  #$37
71C2 9002   6020         BCC  A19
71C4 A900   6030         LDA  #$00
71C6 99EA71 6040 A19     STA  MOSCOW,Y
71C9 88     6050 A18     DEY
71CA 10E2   6060         BPL  LOOPH
            6070 ;
71CC 20D172 6080         JSR  DINGO
71CF 200072 6090         JSR  STALL
71D2 A6C2   6100 Y06     LDX  ARMY
71D4 CA     6110         DEX
71D5 F003   6120         BEQ  Y07
71D7 4CE170 6130         JMP  LOOP32
71DA EE2E06 6140 Y07     INC  TICK
71DD AD2E06 6150         LDA  TICK
71E0 C920   6160         CMP  #$20
71E2 F003   6170         BEQ  Y08
71E4 4CDF70 6180         JMP  LOOP33
            6190 ;
            6200 ;end of movement phase
            6210 ;
71E7 4C9A6E 6220 Y08     JMP  NEWTRN
            6230 ;
```

```
71EA 00      6240 MOSCOW .BYTE 0,0,0,0
71EB 00
71EC 00
71ED 00
             6250 ;
71EE         6260        *=   $7200
7200 A900    6270 STALL  LDA  #$00
7202 48      6280 LOOP79 PHA
7203 68      6290        PLA
7204 48      6300        PHA
7205 68      6310        PLA
7206 48      6320        PHA
7207 68      6330        PLA
7208 6901    6340        ADC  #$01
720A D0F6    6350        BNE  LOOP79
720C 60      6360        RTS
             6370 ;
             6380 ;this is the debugging routine
             6390 ;it can't be reached by any route any longer
             6400 ;
             6410 ;
720D         6420        *=   $7210
7210 A900    6430        LDA  #$00
7212 8D1DD0  6440        STA  $D01D
7215 8D0DD0  6450        STA  $D00D
7218 8D0ED0  6460        STA  $D00E
721B 8D0FD0  6470        STA  $D00F
721E A922    6480        LDA  #$22
7220 8D2F02  6490        STA  $22F
7223 A920    6500        LDA  #$20
7225 8D3002  6510        STA  $230
7228 A9BC    6520        LDA  #$BC
722A 8D3102  6530        STA  $231
722D A940    6540        LDA  #$40
722F 8D0ED4  6550        STA  NMIEN
7232 A90A    6560        LDA  #$0A
7234 8DC502  6570        STA  $2C5
7237 A900    6580        LDA  #$00
7239 8DFF5F  6590        STA  $5FFF
723C 8DC802  6600        STA  $2C8
723F 00      6610        BRK
             6620 ;
             6630 ;
             6640 ;
             6650 ;Subroutine TERR determines what terrain is in a square
             6660 ;
7240         6670        *=   $7240
             6680 ;
7240 204672  6690 TERR   JSR  TERRB
7243 F046    6700        BEQ  LOOKUP
7245 60      6710        RTS
7246 A900    6720 TERRB  LDA  #$00
7248 85C1    6730        STA  MAPPTR+1
724A 85C3    6740        STA  UNITNO
724C A927    6750        LDA  #$27
724E 38      6760        SEC
724F E5CA    6770        SBC  LAT
7251 0A      6780        ASL  A
7252 26C1    6790        ROL  MAPPTR+1
7254 0A      6800        ASL  A
7255 26C1    6810        ROL  MAPPTR+1
7257 0A      6820        ASL  A
7258 26C1    6830        ROL  MAPPTR+1
725A 0A      6840        ASL  A
725B 26C1    6850        ROL  MAPPTR+1
725D 8D2C06  6860        STA  TLO
7260 A4C1    6870        LDY  MAPPTR+1
7262 8C2D06  6880        STY  THI
7265 0A      6890        ASL  A
7266 26C1    6900        ROL  MAPPTR+1
7268 18      6910        CLC
7269 6D2C06  6920        ADC  TLO
726C 85C0    6930        STA  MAPPTR
726E A5C1    6940        LDA  MAPPTR+1
7270 6D2D06  6950        ADC  THI
7273 6965    6960        ADC  #$65
7275 85C1    6970        STA  MAPPTR+1
7277 A92E    6980        LDA  #46
7279 38      6990        SEC
727A E5CB    7000        SBC  LONG
727C A8      7010        TAY
727D B1C0    7020        LDA  (MAPPTR),Y
727F 8D2B06  7030        STA  TRNCOD
7282 293F    7040        AND  #$3F
7284 C93D    7050        CMP  #$3D
7286 F002    7060        BEQ  A80
7288 C93E    7070        CMP  #$3E
728A 60      7080 A80    RTS
             7090 ;
728B AD2B06  7100 LOOKUP LDA  TRNCOD
728E 8D2F06  7110        STA  UNTCOD
7291 29C0    7120        AND  #$C0
7293 A29E    7130        LDX  #$9E
7295 C940    7140        CMP  #$40
7297 D002    7150        BNE  X98
7299 A237    7160        LDX  #$37
729B A5CA    7170 X98    LDA  LAT
729D DD9F54  7180 LOOP30 CMP  CORPSY,X
72A0 F00A    7190        BEQ  MIGHTB
72A2 CA      7200 X97    DEX
72A3 D0F8    7210        BNE  LOOP30
72A5 A9FF    7220        LDA  #$FF
72A7 8DD064  7230        STA  TXTWDW+128
72AA 301C    7240        BMI  MATCH
```

```
72AC A5CB   7250 MIGHTB LDA  LONG
72AE DD0054 7260        CMP  CORPSX,X
72B1 D010   7270        BNE  X99
72B3 BDDD55 7280        LDA  CSTRNG,X
72B6 F00B   7290        BEQ  X99
72B8 BD1B57 7300        LDA  ARRIVE,X
72BB 3006   7310        BMI  X99
72BD C5C9   7320        CMP  TURN
72BF 9007   7330        BCC  MATCH
72C1 F005   7340        BEQ  MATCH
72C3 A5CA   7350 X99    LDA  LAT
72C5 4CA272 7360        JMP  X97
72C8 86C3   7370 MATCH  STX  UNITNO
72CA BD7C56 7380        LDA  SWAP,X
72CD 8D2B06 7390        STA  TRNCOD
72D0 60     7400        RTS
            7410 ;
            7420 ;determines execution time of next move
            7430 ;
72D1 A6C2   7440 DINGO  LDX  ARMY
72D3 BD755D 7450        LDA  HMORDS,X
72D6 D006   7460        BNE  Y00
72D8 A9FF   7470        LDA  #$FF
72DA 9D616D 7480        STA  EXEC,X
72DD 60     7490        RTS
72DE BD0054 7500 Y00    LDA  CORPSX,X
72E1 85CB   7510        STA  LONG
72E3 BD9F54 7520        LDA  CORPSY,X
72E6 85CA   7530        STA  LAT
72E8 204072 7540        JSR  TERR
72EB AD2F06 7550        LDA  UNTCOD
72EE 8D3006 7560        STA  UNTCD1
72F1 A6C2   7570        LDX  ARMY
72F3 BD145E 7580        LDA  WHORDS,X
72F6 4902   7590        EOR  #$02
72F8 2903   7600        AND  #$03
72FA A8     7610        TAY
72FB BD0054 7620        LDA  CORPSX,X
72FE 18     7630        CLC
72FF 79E25F 7640        ADC  XADD,Y
7302 85CB   7650        STA  LONG
7304 BD9F54 7660        LDA  CORPSY,X
7307 18     7670        CLC
7308 79E65F 7680        ADC  YADD,Y
730B 85CA   7690        STA  LAT
730D 204072 7700        JSR  TERR
7310 206973 7710        JSR  TERRTY
7313 AD3006 7720        LDA  UNTCD1
7316 293F   7730        AND  #$3F
7318 A200   7740        LDX  #$00
731A C93D   7750        CMP  #$3D
731C F002   7760        BEQ  Y01        Infantry
731E A20A   7770        LDX  #$0A       armor
7320 8A     7780 Y01    TXA
7321 AE0C06 7790        LDX  MONTH
7324 18     7800        CLC
7325 7DC06C 7810        ADC  SSNCOD-1,X add season index
7328 65CD   7820        ADC  TRNTYP     add terrain index
732A AA     7830        TAX
732B BDCD6C 7840        LDA  TRNTAB,X   get net delay
732E 18     7850        CLC
732F 6D2E06 7860        ADC  TICK
7332 A6C2   7870        LDX  ARMY
7334 9D616D 7880        STA  EXEC,X
7337 A5CD   7890        LDA  TRNTYP
7339 C907   7900        CMP  #$07
733B 902B   7910        BCC  Y02
733D A015   7920        LDY  #$15
733F A5CA   7930 LOOP35 LDA  LAT
7341 D91F6D 7940        CMP  BHY1,Y
7344 D01F   7950        BNE  Y03
7346 A5CB   7960        LDA  LONG
7348 D9096D 7970        CMP  BHX1,Y
734B D018   7980        BNE  Y03
734D A6C2   7990        LDX  ARMY
734F BD0054 8000        LDA  CORPSX,X
7352 D9356D 8010        CMP  BHX2,Y
7355 D00E   8020        BNE  Y03
7357 BD9F54 8030        LDA  CORPSY,X
735A D94B6D 8040        CMP  BHY2,Y
735D D006   8050        BNE  Y03
735F A9FF   8060        LDA  #$FF
7361 9D616D 8070        STA  EXEC,X
7364 60     8080        RTS
7365 88     8090 Y03    DEY
7366 10*>g8100      BPL  LOOP35
7368 60     8110 Y02    RTS
            8120 ;
            8130 ;this subroutine determines the type of terrain
            8140 ;in a square
            8150 ;
7369 A000   8160 TERRTY LDY  #$00
736B AD2B06 8170        LDA  TRNCOD
736E F043   8180        BEQ  DONE
7370 C97F   8190        CMP  #$7F       border?
7372 D004   8200        BNE  Y04
7374 A009   8210        LDY  #$09
7376 D03B   8220        BNE  DONE
7378 C8     8230 Y04    INY
7379 C907   8240        CMP  #$07       mountain?
737B 9036   8250        BCC  DONE
737D C8     8260        INY
737E C94B   8270        CMP  #$4B       city?
7380 9031   8280        BCC  DONE
```

```
7382 C8      8290         INY
7383 C94F    8300         CMP  #$4F       frozen swamp?
7385 902C    8310         BCC  DONE
7387 C8      8320         INY
7388 C969    8330         CMP  #$69       frozen river?
738A 9027    8340         BCC  DONE
738C C8      8350         INY
738D C98F    8360         CMP  #$8F       swamp?
738F 9022    8370         BCC  DONE
7391 C8      8380         INY
7392 C9A4    8390         CMP  #$A4       river?
7394 901D    8400         BCC  DONE
7396 A6CA    8410         LDX  LAT
7398 E00E    8420         CPX  #$0E
739A 9004    8430         BCC  NEXT
739C C9A9    8440         CMP  #$A9
739E 9013    8450         BCC  DONE
73A0 C8      8460 NEXT    INY
73A1 C9BA    8470         CMP  #$BA       coastline?
73A3 900E    8480         BCC  DONE
73A5 E00E    8490         CPX  #$0E
73A7 9004    8500         BCC  NEXT2
73A9 C9BB    8510         CMP  #$BB
73AB 9006    8520         BCC  DONE
73AD C8      8530 NEXT2   INY
73AE C9BD    8540         CMP  #$BD       estuary?
73B0 9001    8550         BCC  DONE
73B2 C8      8560         INY
73B3 84CD    8570 DONE    STY  TRNTYP
73B5 60      8580         RTS
73B6 00      8590 ZPVAL   .BYTE 0,$64,0,0,0,$22,1,$30,2
73B7 64
73B8 00
73B9 00
73BA 00
73BB 22
73BC 01
73BD 30
73BE 02
73BF E0      8600 PSXVAL  .BYTE $E0,0,0,$33,$78,$D6,$10,$27
73C0 00
73C1 00
73C2 33
73C3 78
73C4 D6
73C5 10
73C6 27
73C7 40      8610         .BYTE $40,0,1,15,6,41,0,1
73C8 00
73C9 01
73CA 0F
73CB 06
73CC 29
73CD 00
73CE 01
73CF 58      8620 COLTAB  .BYTE $58,$DC,$2F,0,$6A,$C,$94,$46,$B0
73D0 DC
73D1 2F
73D2 00
73D3 6A
73D4 0C
73D5 94
73D6 46
73D7 B0
73D8 14      8630 MPTS    .BYTE 20,10,10,10
73D9 0A
73DA 0A
73DB 0A
73DC 14      8640 MOSCX   .BYTE 20,33,20,6
73DD 21
73DE 14
73DF 06
73E0 1C      8650 MOSCY   .BYTE 28,36,0,15
73E1 24
73E2 00
73E3 0F
73E4 0A      8660 TXTMSG  ASL  A
73E5 0A      8670         ASL  A
73E6 0A      8680         ASL  A
73E7 0A      8690         ASL  A
73E8 0A      8700         ASL  A
73E9 AA      8710         TAX
73EA A069    8720         LDY  #$69
73EC BD085D  8730 LOOP19  LDA  TXTTBL,X
73EF 38      8740         SEC
73F0 E920    8750         SBC  #$20
73F2 995064  8760         STA  TXTWDW,Y
73F5 C8      8770         INY
73F6 E8      8780         INX
73F7 8A      8790         TXA
73F8 291F    8800         AND  #$1F
73FA D0F0    8810         BNE  LOOP19
73FC 60      8820         RTS
73FD         8830         .END
```

```
                10 ;EFT VERSION 1.8C (COMBAT) 11/30/81 COPYRIGHT CHRIS CRAWFORD 1981
                20 ;
                30 ;Page zero RAM
                40 ;
                50 ;These locations are for the mainline routines
                60 ;
00BE            70 CHUNKX   =    $BE
00BF            80 CHUNKY   =    $BF
00B4            90 CORPS    =    $B4
0000           0100         *=   $C0
00C0           0110 MAPPTR  *=   *+2
00C2           0120 ARMY    *=   *+1
00C3           0130 UNITNO  *=   *+1
00C4           0140 DEFNDR  *=   *+1
00C5           0150 TEMPR   *=   *+1
00C6           0160 TEMPZ   *=   *+1
00C7           0170 ACCLO   *=   *+1
00C8           0180 ACCHI   *=   *+1
00C9           0190 TURN    *=   *+1
00CA           0200 LAT     *=   *+1
00CB           0210 LONG    *=   *+1
00CC           0220 RFR     *=   *+1
00CD           0230 TRNTYP  *=   *+1
00CE           0240 SQVAL   *=   *+1
               0250 ;
               0260 ;
D01F           0270 CONSOL  =    $D01F
D200           0280 AUDF1   =    $D200
D201           0290 AUDC1   =    $D201
D20A           0300 RANDOM  =    $D20A
D40E           0310 NMIEN   =    $D40E
               0320 ;
               0330 ;THESE VALUES ARE USED BY MAINLINE ROUTINE ONLY
               0340 ;
0606           0350 EARTH   =    $606
062B           0360 TRNCOD  =    $62B
00CF           0370         *=   $636
0636           0380 SQX     *=   *+1        adjacent square
0637           0390 SQY     *=   *+1
0638           0400         *=   $68E
068E           0410 DELAY   *=   *+1
068F           0420 HANDCP  *=   *+1
0690           0430 TOTGS   *=   *+1
0691           0440 TOTRS   *=   *+1
0692           0450 OFR     *=   *+1
0693           0460 HOMEDR  *=   *+1
0694           0470 ZOC     *=   *+1
0695           0480 TEMPQ   *=   *+1
0696           0490 LLIM    *=   *+1
0697           0500 VICTRY  *=   *+1
               0510 ;
               0520 ;declarations of routines in other modules
               0530 ;
4D26           0540 INVERT  =    $4D26
7200           0550 STALL   =    $7200
7240           0560 TERR    =    $7240
7246           0570 TERRB   =    $7246
72DE           0580 YOO     =    $72DE
7369           0590 TERRTY  =    $7369
7BB2           0600 DNUMBR  =    $7BB2
799C           0610 JSTP    =    $799C
79C0           0620 DWORDS  =    $79C0
79EF           0630 SWITCH  =    $79EF
79B4           0640 DEFNC   =    $79B4
7BF6           0650 OFFNC   =    $7BF6
7BF2           0660 XINC    =    $7BF2
7BF1           0670 YINC    =    $7BF1
               0680 ;
0698           0690         *=   $5400
5400           0700 CORPSX  *=   *+159      x-coords of all units (pixel frame)
549F           0710 CORPSY  *=   *+159      y-coords of all units (pixel frame)
553E           0720 MSTRNG  *=   *+159      muster strengths
55DD           0730 CSTRNG  *=   *+159      combat strengths
567C           0740 SWAP    *=   *+159      terrain code underneath unit
571B           0750 ARRIVE  *=   *+159      turn of arrival
57BA           0760 WORDS   *=   *+272      various words for messages
58CA           0770 CORPT   *=   *+159      codes for unit types
5969           0780 CORPNO  *=   *+159      ID numbers of units
5A08           0790 HDIGIT  *=   *+256      tables for displaying numbers (hundreds)
5B08           0800 TDIGIT  *=   *+256      tens tables
5C08           0810 ODIGIT  *=   *+256      ones tables
5D08           0820 TXTTBL  *=   *+96       more text
5D68           0830 MONLEN  *=   *+13       table of month lengths
5D75           0840 HMORDS  *=   *+159      how many orders each unit has in queue
5E14           0850 WHORDS  *=   *+159      what the orders are
5EB3           0860 WHORDH  *=   *+159
5F52           0870 BEEPTB  *=   *+4        table of beep tones
5F56           0880 ERRMSG  *=   *+128      table of error messages
5FD6           0890 XOFF    *=   *+4        offsets for moving maltakreuze
5FDA           0900 YOFF    *=   *+4
5FDE           0910 MASKO   *=   *+4        mask values for decoding orders
5FE2           0920 XADD    *=   *+4        offsets for moving arrow
5FE6           0930 YADD    *=   *+4
5FEA           0940 TRTAB   *=   *+13       tree color table
5FF7           0950 MLTKRZ  *=   *+8        maltese cross shape
               0960 ;
               0970 ;RAM from $6000 to $6430 is taken up by
               0980 ;character sets and the display list
               0990 ;
5FFF           1000         *=   $6431
6431           1010 ARRTAB  *=   *+32       arrow shapes
6451           1020         *=   $6450
6450           1030 TXTWDW  *=   $6CB1
```

```
6CB1          1040 STKTAB *=   *+16      a joystick decoding table
6CC1          1050 SSNCOD *=   *+12      season codes
6CCD          1060 TRNTAB *=   *+60      terrain cost tables
6D09          1070 BHX1   *=   *+22      intraversible square pair coordinates
6D1F          1080 BHY1   *=   *+22
6D35          1090 BHX2   *=   *+22
6D4B          1100 BHY2   *=   *+22
6D61          1110 EXEC   *=   *+159     execution times
              1120 ;
6E00          1130        *=   $4ED8
              1140 ;
              1150 ;combat routine
              1160 ;
4ED8 A900     1170        LDA  #$00
4EDA 8D9706   1180        STA  VICTRY    clear victory flag
4EDD A6C2     1190        LDX  ARMY
4EDF E02A     1200        CPX  #$2A      Finns can't attack
4EE1 F004     1210        BEQ  A10
4EE3 E02B     1220        CPX  #$2B
4EE5 D001     1230        BNE  A11
4EE7 60       1240 A10    RTS
4EE8 A4C3     1250 A11    LDY  UNITNO
4EEA 84C4     1260        STY  DEFNDR
4EEC A6C4     1270        LDX  DEFNDR    make combat graphics
4EEE BD7C56   1280        LDA  SWAP,X
4EF1 48       1290        PHA
4EF2 A9FF     1300        LDA  #$FF      solid red square
4EF4 E037     1310        CPX  #$37      Russian unit?
4EF6 B002     1320        BCS  B1
4EF8 A97F     1330        LDA  #$7F      make it white for Germans
4EFA 9D7C56   1340 B1     STA  SWAP,X
4EFD 86B4     1350        STX  CORPS
4EFF BD0054   1360        LDA  CORPSX,X
4F02 85BE     1370        STA  CHUNKX
4F04 BD9F54   1380        LDA  CORPSY,X
4F07 85BF     1390        STA  CHUNKY
4F09 20EF79   1400        JSR  SWITCH
4F0C A008     1410        LDY  #$08
4F0E A28F     1420        LDX  #$8F
4F10 8E01D2   1430 LOOP78 STX  AUDC1
4F13 8C00D2   1440        STY  AUDF1
4F16 200072   1450        JSR  STALL
4F19 98       1460        TYA
4F1A 18       1470        CLC
4F1B 6908     1480        ADC  #$08
4F1D A8       1490        TAY
4F1E CA       1500        DEX
4F1F E07F     1510        CPX  #$7F
4F21 D0ED     1520        BNE  LOOP78
              1530 ;
              1540 ;now replace original unit character
              1550 ;
4F23 20EF79   1560        JSR  SWITCH
4F26 A6C4     1570        LDX  DEFNDR
4F28 68       1580        PLA
4F29 9D7C56   1590        STA  SWAP,X
              1600 ;
              1610 ;
4F2C 206973   1620        JSR  TERRTY    terrain in defender's square
4F2F BEB479   1630        LDX  DEFNC,Y   defensive bonus factor
4F32 B9DD55   1640        LDA  CSTRNG,Y  defender's strength
4F35 4A       1650        LSR  A
4F36 CA       1660 Y15    DEX            adjust for terrain
4F37 F005     1670        BEQ  Y16
4F39 2A       1680        ROL  A
4F3A 90FA     1690        BCC  Y15
4F3C A9FF     1700        LDA  #$FF
              1710 ;
              1720 ;now adjust for defender's motion
              1730 ;
4F3E BE755D   1740 Y16    LDX  HMORDS,Y
4F41 F001     1750        BEQ  DOBATL
4F43 4A       1760        LSR  A
              1770 ;
              1780 ;evaluate defender's strike
              1790 ;
4F44 CD0AD2   1800 DOBATL CMP  RANDOM
4F47 9017     1810        BCC  ATAKR
4F49 A6C2     1820        LDX  ARMY
4F4B DE3E55   1830        DEC  MSTRNG,X
4F4E BDDD55   1840        LDA  CSTRNG,X
4F51 E905     1850        SBC  #$05
4F53 9DDD55   1860        STA  CSTRNG,X
4F56 F002     1870        BEQ  Z28
4F58 B003     1880        BCS  Y24
4F5A 4CAB51   1890 Z28    JMP  DEAD      attacker dies
4F5D 20CE51   1900 Y24    JSR  BRKCHK    attacker lives; does he break?
              1910 ;
              1920 ;evaluate attacker's strike
              1930 ;
4F60 A6C2     1940 ATAKR  LDX  ARMY
4F62 BD0054   1950        LDA  CORPSX,X
4F65 85CB     1960        STA  LONG
4F67 BD9F54   1970        LDA  CORPSY,X
4F6A 85CA     1980        STA  LAT
4F6C 204072   1990        JSR  TERR
4F6F 206973   2000        JSR  TERRTY
4F72 B9F67B   2010        LDA  OFFNC,Y
4F75 A8       2020        TAY
4F76 A6C2     2030        LDX  ARMY
4F78 BDDD55   2040        LDA  CSTRNG,X
4F7B 88       2050        DEY
4F7C F001     2060        BEQ  Y19
4F7E 4A       2070        LSR  A         river attack penalty
```

```
4F7F CD0AD2 2080 Y19   CMP  RANDOM
4F82 9014   2090       BCC  A20
4F84 A6C4   2100       LDX  DEFNDR     attacker strikes defender
4F86 DE3E55 2110       DEC  MSTRNG,X
4F89 BDDD55 2120       LDA  CSTRNG,X
4F8C E905   2130       SBC  #$05
4F8E 9DDD55 2140       STA  CSTRNG,X
4F91 F002   2150       BEQ  Z29
4F93 B006   2160       BCS  Y25
4F95 20AB51 2170 Z29   JSR  DEAD       defender dies
4F98 4C1C50 2180 A20   JMP  ENDCOM
4F9B 20CE51 2190 Y25   JSR  BRKCHK     does defender break?
4F9E 90F8   2200       BCC  A20
4FA0 A4C2   2210       LDY  ARMY
4FA2 B9145E 2220       LDA  WHORDS,Y
4FA5 2903   2230       AND  #$03
4FA7 A8     2240       TAY             first retreat priority : away from attack
4FA8 202250 2250       JSR  RETRET
4FAB 9054   2260       BCC  VICCOM     defender died
4FAD F030   2270       BEQ  Y27        defender may retreat
4FAF A001   2280       LDY  #$01       second priority: east/west
4FB1 E037   2290       CPX  #$37
4FB3 B002   2300       BCS  Y28
4FB5 A003   2310       LDY  #$03
4FB7 202250 2320 Y28   JSR  RETRET
4FBA 9045   2330       BCC  VICCOM
4FBC F021   2340       BEQ  Y27
4FBE A002   2350       LDY  #$02       third priority: north
4FC0 202250 2360       JSR  RETRET
4FC3 903C   2370       BCC  VICCOM
4FC5 F018   2380       BEQ  Y27
4FC7 A000   2390       LDY  #$00       fourth priority: south
4FC9 202250 2400       JSR  RETRET
4FCC 9033   2410       BCC  VICCOM
4FCE F00F   2420       BEQ  Y27
4FD0 A003   2430       LDY  #$03       last priority: west/east
4FD2 E037   2440       CPX  #$37
4FD4 B002   2450       BCS  Y26
4FD6 A001   2460       LDY  #$01
4FD8 202250 2470 Y26   JSR  RETRET
4FDB 9024   2480       BCC  VICCOM
4FDD D03D   2490       BNE  ENDCOM
4FDF 86B4   2500 Y27   STX  CORPS      retreat the defender
4FE1 BD0054 2510       LDA  CORPSX,X
4FE4 85BE   2520       STA  CHUNKX
4FE6 BD9F54 2530       LDA  CORPSY,X
4FE9 85BF   2540       STA  CHUNKY
4FEB 20EF79 2550       JSR  SWITCH
4FEE A6B4   2560       LDX  CORPS
4FF0 A5CA   2570       LDA  LAT
4FF2 9D9F54 2580       STA  CORPSY,X
4FF5 85BF   2590       STA  CHUNKY
4FF7 A5CB   2600       LDA  LONG
4FF9 9D0054 2610       STA  CORPSX,X
4FFC 85BE   2620       STA  CHUNKX
4FFE 20EF79 2630       JSR  SWITCH
5001 A6C2   2640 VICCOM LDX  ARMY
5003 86B4   2650       STX  CORPS
5005 BD0054 2660       LDA  CORPSX,X
5008 85BE   2670       STA  CHUNKX
500A BD9F54 2680       LDA  CORPSY,X
500D 85BF   2690       STA  CHUNKY
500F A5C7   2700       LDA  ACCLO      defender's coordinates
5011 85CB   2710       STA  LONG
5013 A5C8   2720       LDA  ACCHI
5015 85CA   2730       STA  LAT
5017 A9FF   2740       LDA  #$FF
5019 8D9706 2750       STA  VICTRY
501C A6C2   2760 ENDCOM LDX  ARMY
501E FE616D 2770       INC  EXEC,X
5021 60     2780       RTS
            2790 ;
            2800 ;Subroutines for combat
            2810 ;input: X = ID # of defender. Y = proposed DIR of retreat
            2820 ;output: C bit set if defender lives, clear if dies
            2830 ;Z bit set if retreat open, clear if blocked
            2840 ;
5022 BD0054 2850 RETRET LDA  CORPSX,X
5025 18     2860       CLC
5026 79F27B 2870       ADC  XINC,Y
5029 85CB   2880       STA  LONG
502B BD9F54 2890       LDA  CORPSY,X
502E 18     2900       CLC
502F 79F17B 2910       ADC  YINC,Y
5032 85CA   2920       STA  LAT
5034 204072 2930       JSR  TERR       examine terrain
5037 206973 2940       JSR  TERRTY
503A A6C4   2950       LDX  DEFNDR
503C A5C3   2960       LDA  UNITNO     anybody in this square?
503E D03D   2970       BNE  Y22
5040 A5CD   2980       LDA  TRNTYP     no
            2990 ;
            3000 ;check for bad ocean crossings
            3010 ;
5042 C907   3020       CMP  #$07       coastline?
5044 9027   3030       BCC  Y41
5046 C909   3040       CMP  #$09
5048 F033   3050       BEQ  Y22
504A A015   3060       LDY  #$15
504C A5CA   3070 LOOP42 LDA  LAT
504E D91F6D 3080       CMP  BHY1,Y
5051 D017   3090       BNE  Y43
5053 A5CB   3100       LDA  LONG
5055 D9096D 3110       CMP  BHX1,Y
```

```
5058 D010   3120        BNE  Y43
505A BD0054 3130        LDA  CORPSX,X
505D D9356D 3140        CMP  BHX2,Y
5060 D008   3150        BNE  Y43
5062 BD9F54 3160        LDA  CORPSY,X
5065 D94B6D 3170        CMP  BHY2,Y
5068 F013   3180        BEQ  Y22
506A 88     3190 Y43    DEY
506B 10DF   3200        BPL  LOOP42
            3210 ;
            3220 ;any blocking ZOC's?
            3230 ;
506D 204051 3240 Y41    JSR  CHKZOC
5070 A6C4   3250        LDX  DEFNDR
5072 AD9406 3260        LDA  ZOC
5075 C902   3270        CMP  #$02
5077 B004   3280        BCS  Y22        no retreat into ZOC
5079 A900   3290        LDA  #$00       retreat is possible
507B 38     3300        SEC
507C 60     3310        RTS
507D BDDD55 3320 Y22    LDA  CSTRNG,X   retreat not possible,extract penalty
5080 38     3330        SEC
5081 E905   3340        SBC  #$05
5083 9DDD55 3350        STA  CSTRNG,X
5086 F002   3360        BEQ  Z27
5088 B004   3370        BCS  Y23
508A 20AB51 3380 Z27    JSR  DEAD
508D 18     3390        CLC
508E A9FF   3400 Y23    LDA  #$FF
5090 60     3410        RTS
            3420 ;
            3430 ;supply evaluation routine
            3440 ;
5091 BD1B57 3450        LDA  ARRIVE,X
5094 C5C9   3460        CMP  TURN
5096 F003   3470        BEQ  Z86
5098 9001   3480        BCC  Z86
509A 60     3490        RTS
509B A918   3500 Z86    LDA  #$18
509D E037   3510        CPX  #$37
509F B01B   3520        BCS  A13
50A1 A918   3530        LDA  #$18
50A3 AC0606 3540        LDY  EARTH
50A6 C002   3550        CPY  #$02       mud?
50A8 F06B   3560        BEQ  A12
50AA C00A   3570        CPY  #$0A       snow?
50AC D00E   3580        BNE  A13
50AE BD0054 3590        LDA  CORPSX,X   this discourages gung-ho corps
50B1 0A     3600        ASL  A          double distance
50B2 0A     3610        ASL  A
50B3 694A   3620        ADC  #$4A
50B5 CD0AD2 3630        CMP  RANDOM
50B8 905B   3640        BCC  A12
50BA A910   3650        LDA  #$10       harder to get supplies in winter
50BC 85C7   3660 A13    STA  ACCLO
50BE A001   3670        LDY  #$01       Russians go east
50C0 E037   3680        CPX  #$37
50C2 B002   3690        BCS  Z80
50C4 A003   3700        LDY  #$03       Germans go west
50C6 8C9306 3710 Z80    STY  HOMEDR
50C9 BD0054 3720        LDA  CORPSX,X
50CC 85CB   3730        STA  LONG
50CE BD9F54 3740        LDA  CORPSY,X
50D1 85CA   3750        STA  LAT
50D3 A900   3760        ]> #$00
50D5 85CC   3770        STA  RFR
50D7 A5CB   3780 LOOP91 LDA  LONG
50D9 8D3606 3790        STA  SQX
50DC A5CA   3800        LDA  LAT
50DE 8D3706 3810        STA  SQY
50E1 AD3606 3820 LOOP90 LDA  SQX
50E4 18     3830        CLC
50E5 79F27B 3840        ADC  XINC,Y
50E8 85CB   3850        STA  LONG
50EA AD3706 3860        LDA  SQY
50ED 18     3870        CLC
50EE 79F17B 3880        ADC  YINC,Y
50F1 85CA   3890        STA  LAT
50F3 204051 3900        JSR  CHKZOC
50F6 E037   3910        CPX  #$37
50F8 900A   3920        BCC  A80
50FA 204672 3930        JSR  TERRB
50FD AD2B06 3940        LDA  TRNCOD
5100 C9BF   3950        CMP  #$BF
5102 F009   3960        BEQ  A77
5104 AD9406 3970 A80    LDA  ZOC
5107 C902   3980        CMP  #$02
5109 901C   3990        BCC  Z81
510B E6CC   4000        INC  RFR
510D E6CC   4010 A77    INC  RFR
510F A5CC   4020        LDA  RFR
5111 C5C7   4030        CMP  ACCLO
90 5113 D009   4040        BNE  Z84     BCC
5115 5EDD55 4050 A12    LSR  CSTRNG,X
5118 D003   4060        BNE  A50
511A 4CAB51 4070        JMP  DEAD
511D 60     4080 A50    RTS
511E AD0AD2 4090 Z84    LDA  RANDOM
5121 2902   4100        AND  #$02
5123 A8     4110        TAY
5124 4CE150 4120        JMP  LOOP90
5127 AC9306 4130 Z81    LDY  HOMEDR
512A A5CB   4140        LDA  LONG
512C C001   4150        CPY  #$01
```

```
512E D00B   4160         BNE  Z85
5130 C9FF   4170         CMP  #$FF
5132 D0A3   4180         BNE  LOOP91
5134 FE3E55 4190         INC  MSTRNG,X  Russian replacements
5137 FE3E55 4200         INC  MSTRNG,X
513A 60     4210         RTS
513B C92E   4220 Z85     CMP  #$2E
513D D098   4230         BNE  LOOP91
513F 60     4240         RTS
            4250 ;
            4260 ;routine to check for zone of control
            4270 ;
5140 A900   4280 CHKZOC  LDA  #$00
5142 8D9406 4290         STA  ZOC
5145 A940   4300         LDA  #$40
5147 E037   4310         CPX  #$37
5149 B002   4320         BCS  A70
514B A9C0   4330         LDA  #$C0
514D 85C5   4340 A70     STA  TEMPR
514F 204672 4350         JSR  TERRB
5152 D01E   4360         BNE  A74
5154 AD2B06 4370         LDA  TRNCOD
5157 29C0   4380         AND  #$C0
5159 C5C5   4390         CMP  TEMPR
515B F00F   4400         BEQ  A71
515D BD0054 4410         LDA  CORPSX,X
5160 C5CB   4420         CMP  LONG
5162 D007   4430         BNE  A79
5164 BD9F54 4440         LDA  CORPSY,X
5167 C5CA   4450         CMP  LAT
5169 F007   4460         BEQ  A74
516B 60     4470 A79     RTS
516C A902   4480 A71     LDA  #$02
516E 8D9406 4490         STA  ZOC
5171 60     4500         RTS
5172 A207   4510 A74     LDX  #$07
5174 BCAC79 4520 LOOPQ   LDY  JSTP+16,X
5177 A5CB   4530         LDA  LONG
5179 18     4540         CLC
517A 79F27B 4550         ADC  XINC,Y
517D 85CB   4560         STA  LONG
517F A5CA   4570         LDA  LAT
5181 18     4580         CLC
5182 79F17B 4590         ADC  YINC,Y
5185 85CA   4600         STA  LAT
5187 204672 4610         JSR  TERRB
518A D015   4620         BNE  A75
518C AD2B06 4630         LDA  TRNCOD
518F 29C0   4640         AND  #$C0
5191 C5C5   4650         CMP  TEMPR
5193 D00C   4660         BNE  A75
5195 8A     4670         TXA
5196 2901   4680         AND  #$01
5198 18     4690         CLC
5199 6901   4700         ADC  #$01
519B 6D9406 4710         ADC  ZOC
519E 8D9406 4720         STA  ZOC
51A1 CA     4730 A75     DEX
51A2 10D0   4740         BPL  LOOPQ
51A4 C6CA   4750         DEC  LAT
51A6 C6CB   4760         DEC  LONG
51A8 A6C2   4770         LDX  ARMY
51AA 60     4780         RTS
            4790 ;
            4800 ;
51AB A900   4810 DEAD    LDA  #$00
51AD 9D3E55 4820         STA  MSTRNG,X
51B0 9DDD55 4830         STA  CSTRNG,X
51B3 9D755D 4840         STA  HMORDS,X
51B6 A9FF   4850         LDA  #$FF
51B8 9D616D 4860         STA  EXEC,X
51BB 9D1B57 4870         STA  ARRIVE,X
51BE 86B4   4880         STX  CORPS
51C0 BD0054 4890         LDA  CORPSX,X
51C3 85BE   4900         STA  CHUNKX
51C5 BD9F54 4910         LDA  CORPSY,X
51C8 85BF   4920         STA  CHUNKY
51CA 20EF79 4930         JSR  SWITCH
51CD 60     4940         RTS
            4950 ;
            4960 ;Subroutine BRKCHK evaluates whether a unit under attack breaks
            4970 ;
51CE E037   4980 BRKCHK  CPX  #$37
51D0 B00E   4990         BCS  WEAKLG
51D2 BDCA58 5000         LDA  CORPT,X
51D5 29F0   5010         AND  #$F0
51D7 D007   5020         BNE  WEAKLG
51D9 BD3E55 5030         LDA  MSTRNG,X
51DC 4A     5040         LSR  A
51DD 4CEE51 5050         JMP  Y40
51E0 BD3E55 5060 WEAKLG  LDA  MSTRNG,X
51E3 4A     5070         LSR  A
51E4 4A     5080         LSR  A
51E5 4A     5090         LSR  A
51E6 85C5   5100         STA  TEMPR
51E8 BD3E55 5110         LDA  MSTRNG,X
51EB 38     5120         SEC
51EC E5C5   5130         SBC  TEMPR
51EE DDDD55 5140 Y40     CMP  CSTRNG,X
51F1 900A   5150         BCC  A30
51F3 A9FF   5160         LDA  #$FF
51F5 9D616D 5170         STA  EXEC,X
51F8 A900   5180         LDA  #$00
51FA 9D755D 5190         STA  HMORDS,X
```

```
51FD 60     5200 A30    RTS
            5210 ;
51FE        5220        .END
```

```
                10 ;EFT VERSION 1.8T (THINKING) 11/30/81 COPYRIGHT CHRIS CRAWFORD 1981
                20 ;
                30 ;Page zero RAM
                40 ;
                50 ;These locations are for the mainline routines
                60 ;
00BE            70 CHUNKX   =    $BE
00BF            80 CHUNKY   =    $BF
00B4            90 CORPS    =    $B4
0000           100          *=   $C0
00C0           110 MAPPTR   *=   *+2
00C2           120 ARMY     *=   *+1
00C3           130 UNITNO   *=   *+1
00C4           140 DEFNDR   *=   *+1
00C5           150 TEMPR    *=   *+1
00C6           160 TEMPZ    *=   *+1
00C7           170 ACCLO    *=   *+1
00C8           180 ACCHI    *=   *+1
00C9           190 TURN     *=   *+1
00CA           200 LAT      *=   *+1
00CB           210 LONG     *=   *+1
00CC           220 RFR      *=   *+1
00CD           230 TRNTYP   *=   *+1
00CE           240 SQVAL    *=   *+1
               250 ;
               260 ;
D010           270 TRIGO    =    $D010
D01F           280 CONSOL   =    $D01F
D200           290 AUDF1    =    $D200
D201           300 AUDC1    =    $D201
D20A           310 RANDOM   =    $D20A
D40E           320 NMIEN    =    $D40E
E45C           330 SETVBV   =    $E45C
               340 ;
               350 ;THESE VALUES ARE USED BY MAINLINE ROUTINE ONLY
               360 ;
00CF           370          *=   $605
0605           380 TRCOLR   *=   *+1
0606           390 EARTH    *=   *+1
0607           400 ICELAT   *=   *+1
0608           410 SEASN1   *=   *+1
0609           420 SEASN2   *=   *+1
060A           430 SEASN3   *=   *+1
060B           440 DAY      *=   *+1
060C           450 MONTH    *=   *+1
060D           460 YEAR     *=   *+1
060E           470          *=   $62A
062A           480 OLDLAT   *=   *+1
062B           490 TRNCOD   *=   *+1
062C           500 TLO      *=   *+1
062D           510 THI      *=   *+1
062E           520 TICK     *=   *+1
062F           530 UNTCOD   *=   *+1
0630           540 UNTCD1   *=   *+1
0631           550 BVAL     *=   *+1        best value
0632           560 BONE     *=   *+1        best index
0633           570 DIR      *=   *+1        direction
0634           580 TARGX    *=   *+1        square under consideration
0635           590 TARGY    *=   *+1
0636           600 SQX      *=   *+1        adjacent square
0637           610 SQY      *=   *+1
0638           620 JCNT     *=   *+1        counter for adjacent squares
0639           630 LINCOD   *=   *+1        code value of line configuration
063A           640 NBVAL    *=   *+1        another best value
063B           650 RORD1    *=   *+1        Russian orders
063C           660 RORD2    *=   *+1
063D           670 HDIR     *=   *+1        horizontal direction
063E           680 VDIR     *=   *+1        vertical direction
063F           690 LDIR     *=   *+1        larger direction
0640           700 SDIR     *=   *+1        smaller direction
0641           710 HRNGE    *=   *+1        horizontal range
0642           720 VRNGE    *=   *+1        vertical range
0643           730 LRNGE    *=   *+1        larger range
0644           740 SRNGE    *=   *+1        smaller range
0645           750 CHRIS    *=   *+1        midway counter
0646           760 RANGE    *=   *+1        just that
0647           770 RCNT     *=   *+1        counter for Russian orders
0648           780 SECDIR   *=   *+1        secondary direction
0649           790 POTATO   *=   *+1        a stupid temporary
064A           800 BAKARR   *=   *+25
0663           810 LINARR   *=   *+25
067C           820 IFR0     *=   *+1
067D           830 IFR1     *=   *+1
067E           840 IFR2     *=   *+1
067F           850 IFR3     *=   *+1
0680           860 XLOC     *=   *+1
0681           870 YLOC     *=   *+1
0682           880 TEMPX    *=   *+1
0683           890 TEMPY    *=   *+1
0684           900 LV       *=   *+5
0689           910 LPTS     *=   *+1
068A           920 COLUM    *=   *+1
068B           930 OCOLUM   *=   *+1
068C           940 IFRHI    *=   *+1
068D           950 PASSCT   *=   *+1
068E           960 DELAY    *=   *+1
068F           970 HANDCP   *=   *+1
0690           980 TOTGS    *=   *+1
0691           990 TOTRS    *=   *+1
0692          1000 OFR      *=   *+1
              1010 ;
              1020 ;declarations of routines in other modules
79B4          1030 DEFNC    =    $79B4
```

```
7A78          1040 ROTARR =    $7A78
7A91          1050 OBJX   =    $7A91
799C          1060 JSTP   =    $799C
0693          1070        *=   $7BD9
7BD9          1080 NDX    *=   *+24
7BF1          1090 YINC   *=   *+1
7BF2          1100 XINC   *=   *+4
7BF6          1110 OFFNC  *=   *+10
5398          1120 OBJY   =    $5398
0698          1130 IFR    =    $698
7240          1140 TERR   =    $7240
72DE          1150 YOO    =    $72DE
              1160 ;
7C00          1170        *=   $5400
5400          1180 CORPSX *=   *+159     x-coords of all units (pixel frame)
549F          1190 CORPSY *=   *+159     y-coords of all units (pixel frame)
553E          1200 MSTRNG *=   *+159     muster strengths
55DD          1210 CSTRNG *=   *+159     combat strengths
567C          1220 SWAP   *=   *+159     terrain code underneath unit
571B          1230 ARRIVE *=   *+159     turn of arrival
57BA          1240 WORDS  *=   *+272     various words for messages
58CA          1250 CORPT  *=   *+159     codes for unit types
5969          1260 CORPNO *=   *+159     ID numbers of units
5A08          1270 HDIGIT *=   *+256     tables for displaying numbers (hundreds)
5B08          1280 TDIGIT *=   *+256     tens tables
5C08          1290 ODIGIT *=   *+256     ones tables
5D08          1300 TXTTBL *=   *+96      more text
5D68          1310 MONLEN *=   *+13      table of month lengths
5D75          1320 HMORDS *=   *+159     how many orders each unit has in queue
5E14          1330 WHORDS *=   *+159     what the orders are
5EB3          1340 WHORDH *=   *+159
5F52          1350 BEEPTB *=   *+4       table of beep tones
5F56          1360 ERRMSG *=   *+128     table of error messages
5FD6          1370 XOFF   *=   *+4       offsets for moving maltakreuze
5FDA          1380 YOFF   *=   *+4
5FDE          1390 MASKO  *=   *+4       mask values for decoding orders
5FE2          1400 XADD   *=   *+4       offsets for moving arrow
5FE6          1410 YADD   *=   *+4
5FEA          1420 TRTAB  *=   *+13      tree color table
5FF7          1430 MLTKRZ *=   *+8       maltese cross shape
              1440 ;
              1450 ;RAM from $6000 to $6430 is taken up by
              1460 ;character sets and the display list
              1470 ;
5FFF          1480        *=   $6431
6431          1490 ARRTAB *=   *+32      arrow shapes
6451          1500        *=   $6450
6450          1510 TXTWDW *=   $6CB1
6CB1          1520 STKTAB *=   *+16      a joystick decoding table
6CC1          1530 SSNCOD *=   *+12      season codes
6CCD          1540 TRNTAB *=   *+60      terrain cost tables
6D09          1550 BHX1   *=   *+22      intraversible square pair coordinates
6D1F          1560 BHY1   *=   *+22
6D35          1570 BHX2   *=   *+22
6D4B          1580 BHY2   *=   *+22
6D61          1590 EXEC   *=   *+159     execution times
              1600 ;
              1610 ;
              1620 ;Russian artificial intelligence routine
              1630 ;
6E00          1640        *=   $4700
              1650 ;
              1660 ;initialization loop
              1670 ;
4700 A201     1680        LDX  #$01
4702 85C5     1690        STA  TEMPR
4704 8D9106   1700        STA  TOTRS
4707 8D9006   1710        STA  TOTGS
470A A09E     1720        LDY  #$9E
470C B91B57   1730 LOOP80 LDA  ARRIVE,Y
470F C5C9     1740        CMP  TURN
4711 B00D     1750        BCS  Z50
4713 A5C5     1760        LDA  TEMPR
4715 18       1770        CLC
4716 79DD55   1780        ADC  CSTRNG,Y
4719 85C5     1790        STA  TEMPR
471B 9003     1800        BCC  Z50
471D FE9006   1810        INC  TOTGS,X
4720 88       1820 Z50    DEY
4721 C037     1830        CPY  #$37
4723 B0E7     1840        BCS  LOOP80
4725 A200     1850        LDX  #$00
4727 C000     1860        CPY  #$00
4729 D0E1     1870        BNE  LOOP80
              1880 ;
              1890 ;now shift values 4 places right
              1900 ;
472B AD9106   1910        LDA  TOTRS
472E 85C5     1920        STA  TEMPR
4730 AD9006   1930        LDA  TOTGS
4733 A204     1940        LDX  #$04
4735 0A       1950 LOOP81 ASL  A
4736 9008     1960        BCC  Z51
4738 6A       1970        ROR  A
4739 46C5     1980 LOOP82 LSR  TEMPR
473B CA       1990        DEX
473C D0FB     2000        BNE  LOOP82
473E F003     2010        BEQ  Z52
4740 CA       2020 Z51    DEX
4741 D0F2     2030        BNE  LOOP81
              2040 ;
              2050 ;now calculate overall force ratio
              2060 ;
4743 A0FF     2070 Z52    LDY  #$FF
```

```
4745 A6C5   2080         LDX  TEMPR
4747 F006   2090         BEQ  Z53
4749 38     2100         SEC
474A C8     2110 LOOP83  INY
474B E5C5   2120         SBC  TEMPR
474D B0FB   2130         BCS  LOOP83
474F 8C9206 2140 Z53     STY  OFR
            2150 ;
            2160 ;now calculate individual force ratios
            2170 ;
4752 A29E   2180         LDX  #$9E
4754 86C2   2190 LOOP50  STX  ARMY
4756 BD1B57 2200         LDA  ARRIVE,X
4759 C5C9   2210         CMP  TURN
475B B00F   2220         BCS  Y44
475D 20234C 2230         JSR  CALIFR
4760 BD0054 2240         LDA  CORPSX,X
4763 9D5A7A 2250         STA  OBJX-55,X
4766 BD9F54 2260         LDA  CORPSY,X
4769 9D6153 2270         STA  OBJY-55,X
476C CA     2280 Y44     DEX
476D E037   2290         CPX  #$37
476F B0E3   2300         BCS  LOOP50
            2310 ;
            2320 ;here begins the main loop
            2330 ;
4771 A29E   2340 MLOOP   LDX  #$9E       outer loop for entire Russian army
4773 86C2   2350 LOOP51  STX  ARMY       inner loop for individual armies
4775 BD1B57 2360         LDA  ARRIVE,X
4778 C5C9   2370         CMP  TURN
477A 9003   2380         BCC  Z26
477C 4C114B 2390 Z54     JMP  TOGSCN
477F BDCA58 2400 Z26     LDA  CORPT,X
4782 C904   2410         CMP  #$04
4784 F0F6   2420         BEQ  Z54
4786 AD9206 2430         LDA  OFR        is army near the front?
4789 4A     2440         LSR  A
478A DD6106 2450         CMP  IFR-55,X
478D D056   2460         BNE  Y51        yes
478F 8D3106 2470         STA  BVAL       no, treat as reinforcement
            2480 ;
            2490 ;find nearby beleaguered army
            2500 ;
4792 A09E   2510         LDY  #$9E
4794 B91B57 2520 LOOP52  LDA  ARRIVE,Y
4797 C5C9   2530         CMP  TURN
4799 B033   2540         BCS  Y54
479B B90054 2550         LDA  CORPSX,Y
479E 38     2560         SEC
479F FD0054 2570         SBC  CORPSX,X
47A2 20304D 2580         JSR  INVERT
47A5 85C5   2590         STA  TEMPR
47A7 B99F54 2600         LDA  CORPSY,Y
47AA 38     2610         SEC
47AB FD9F54 2620         SBC  CORPSY,X
47AE 20304D 2630         JSR  INVERT
47B1 18     2640         CLC
47B2 65C5   2650         ADC  TEMPR
47B4 4A     2660         LSR  A
47B5 4A     2670         LSR  A
47B6 4A     2680         LSR  A
47B7 B015   2690         BCS  Y54
47B9 85C5   2700         STA  TEMPR
47BB B96106 2710         LDA  IFR-55,Y
47BE 38     2720         SEC
47BF E5C5   2730         SBC  TEMPR
47C1 900B   2740         BCC  Y54        no good using nearby armies
47C3 CD3106 2750         CMP  BVAL
47C6 9006   2760         BCC  Y54
47C8 8D3106 2770         STA  BVAL
47CB 8C3206 2780         STY  BONE
47CE 88     2790 Y54     DEY
47CF C037   2800         CPY  #$37
47D1 B0C1   2810         BCS  LOOP52
47D3 AC3206 2820         LDY  BONE       beleagueredest army
47D6 B90054 2830         LDA  CORPSX,Y
47D9 9D5A7A 2840         STA  OBJX-55,X
47DC B99F54 2850         LDA  CORPSY,Y
47DF 9D6153 2860         STA  OBJY-55,X
47E2 4C114B 2870         JMP  TOGSCN
            2880 ;
            2890 ;front line armies
            2900 ;
47E5 A9FF   2910 Y51     LDA  #$FF
47E7 8D3306 2920         STA  DIR        a direction of $FF means 'stay put'
47EA 8D3206 2930         STA  BONE
47ED A900   2940         LDA  #$00
47EF 8D3106 2950         STA  BVAL
            2960 ;
            2970 ;ad hoc logic for surrounded people
            2980 ;
47F2 BD694D 2990         LDA  IFRE-55,X
47F5 C910   3000         CMP  #$10
47F7 B009   3010         BCS  Z55
47F9 BD3E55 3020         LDA  MSTRNG,X
47FC 4A     3030         LSR  A
47FD DDDD55 3040         CMP  CSTRNG,X   out of supply?
4800 9010   3050         BCC  DRLOOP
4802 BD0054 3060 Z55     LDA  CORPSX,X   head due east!
4805 38     3070         SEC
4806 E905   3080         SBC  #$05
4808 B002   3090         BCS  Z96
480A A900   3100         LDA  #$00
480C 9D5A7A 3110 Z96     STA  OBJX-55,X
```

```
480F 4C114B 3120         JMP  TOGSCN
4812 BD5A7A 3130 DRLOOP  LDA  OBJX-55,X
4815 AC3306 3140         LDY  DIR
4818 3004   3150         BMI  Y55
481A 18     3160         CLC
481B 79F27B 3170         ADC  XINC,Y
481E 8D3406 3180 Y55     STA  TARGX
4821 BD6153 3190         LDA  OBJY-55,X
4824 AC3306 3200         LDY  DIR
4827 3004   3210         BMI  Y56
4829 18     3220         CLC
482A 79F17B 3230         ADC  YINC,Y
482D 8D3506 3240 Y56     STA  TARGY
4830 A900   3250         LDA  #$00
4832 85CE   3260         STA  SQVAL
4834 AD3306 3270         LDA  DIR
4837 3010   3280         BMI  Y57
4839 9D145E 3290         STA  WHORDS,X
483C 20DE72 3300         JSR  Y00
483F A4C2   3310         LDY  ARMY
4841 B9616D 3320         LDA  EXEC,Y     Is square accessible?
4844 1003   3330         BPL  Y57        yes
4846 4CD64A 3340         JMP  EVALSQ     no, skip this square
            3350 ;
            3360 ;now fill in the direct line array
            3370 ;
4849 A900   3380 Y57     LDA  #$00
484B 8D3906 3390         STA  LINCOD
484E AD3406 3400         LDA  TARGX
4851 8D3606 3410         STA  SQX
4854 AD3506 3420         LDA  TARGY
4857 8D3706 3430         STA  SQY
485A A017   3440         LDY  #$17
485C 8C3806 3450 LOOP56  STY  JCNT
485F B99C79 3460         LDA  JSTP,Y
4862 A8     3470         TAY
4863 AD3606 3480         LDA  SQX
4866 18     3490         CLC
4867 79F27B 3500         ADC  XINC,Y
486A 8D3606 3510         STA  SQX
486D AD3706 3520         LDA  SQY
4870 18     3530         CLC
4871 79F17B 3540         ADC  YINC,Y
4874 8D3706 3550         STA  SQY
            3560 ;
4877 A29E   3570         LDX  #$9E
4879 BD1B57 3580 LOOP55  LDA  ARRIVE,X
487C C5C9   3590         CMP  TURN
487E F002   3600         BEQ  Z25
4880 B019   3610         BCS  Y58
4882 BD5A7A 3620 Z25     LDA  OBJX-55,X
4885 CD3606 3630         CMP  SQX
4888 D011   3640         BNE  Y58
488A BD6153 3650         LDA  OBJY-55,X
488D CD3706 3660         CMP  SQY
4890 D009   3670         BNE  Y58
4892 E4C2   3680         CPX  ARMY
4894 F00A   3690         BEQ  Y31
4896 BD3E55 3700         LDA  MSTRNG,X
4899 D007   3710         BNE  Y59
489B CA     3720 Y58     DEX
489C E037   3730         CPX  #$37
489E B0D9   3740         BCS  LOOP55
48A0 A900   3750 Y31     LDA  #$00
48A2 AC3806 3760 Y59     LDY  JCNT
48A5 BED97B 3770         LDX  NDX,Y
48A8 9D6306 3780         STA  LINARR,X
48AB 88     3790         DEY
48AC 10AE   3800         BPL  LOOP56     D>      3810 ;
48AE A6C2   3820         LDX  ARMY
48B0 BD3E55 3830         LDA  MSTRNG,X
48B3 8D6F06 3840         STA  LINARR+12
48B6 A900   3850         LDA  #$00
48B8 85C7   3860         STA  ACCLO
48BA 85C8   3870         STA  ACCHI
48BC 8D4806 3880         STA  SECDIR
            3890 ;
            3900 ;build LV array
            3910 ;
48BF A200   3920 Y88     LDX  #$00
48C1 8E4906 3930         STX  POTATO
48C4 A000   3940 Y92     LDY  #$00
48C6 BD6306 3950 Y90     LDA  LINARR,X
48C9 D006   3960         BNE  Y89
48CB E8     3970         INX
48CC C8     3980         INY
48CD C005   3990         CPY  #$05
48CF D0F5   4000         BNE  Y90
48D1 AE4906 4010 Y89     LDX  POTATO
48D4 98     4020         TYA
48D5 9D8406 4030         STA  LV,X
48D8 E8     4040         INX
48D9 8E4906 4050         STX  POTATO
48DC E001   4060         CPX  #$01
48DE D004   4070         BNE  Y91
48E0 A205   4080         LDX  #$05
48E2 D0E0   4090         BNE  Y92
48E4 E002   4100 Y91     CPX  #$02
48E6 D004   4110         BNE  Y93
48E8 A20A   4120         LDX  #$0A
48EA D0D8   4130         BNE  Y92
            4140 ;
48EC E003   4150 Y93     CPX  #$03
```

```
48EE D004   4160          BNE  Z40
48F0 A20F   4170          LDX  #$0F
48F2 D0D0   4180          BNE  Y92
48F4 E004   4190 Z40      CPX  #$04
48F6 D004   4200          BNE  Z41
48F8 A214   4210          LDX  #$14
48FA D0C8   4220          BNE  Y92
            4230 ;
48FC A900   4240 Z41      LDA  #$00
48FE A004   4250          LDY  #$04
4900 BE8406 4260 LOOP76   LDX  LV,Y
4903 E005   4270          CPX  #$05
4905 F003   4280          BEQ  Z42
4907 18     4290          CLC
4908 6928   4300          ADC  #$28
490A 88     4310 Z42      DEY
490B 10F3   4320          BPL  LOOP76
            4330 ;
            4340 ;now add bonus if central column is otherwise empty
            4350 ;
490D AC6D06 4360          LDY  LINARR+10
4910 D012   4370          BNE  Y95
4912 AC6E06 4380          LDY  LINARR+11
4915 D00D   4390          BNE  Y95
4917 AC7006 4400          LDY  LINARR+13
491A D008   4410          BNE  Y95
491C AC7106 4420          LDY  LINARR+14
491F D003   4430          BNE  Y95
4921 18     4440          CLC
4922 6930   4450          ADC  #$30
4924 8D8906 4460 Y95      STA  LPTS
            4470 ;
            4480 ;now evaluate blocking penalty
            4490 ;
4927 A200   4500          LDX  #$00
4929 BD8406 4510 LOOP72   LDA  LV,X
492C C904   4520          CMP  #$04
492E B01F   4530          BCS  Y96
4930 85C5   4540          STA  TEMPR
4932 86C6   4550          STX  TEMPZ
4934 8A     4560          TXA
4935 0A     4570          ASL  A
4936 0A     4580          ASL  A
4937 65C6   4590          ADC  TEMPZ
4939 65C5   4600          ADC  TEMPR
493B A8     4610          TAY
493C C8     4620          INY
493D B96306 4630          LDA  LINARR,Y
4940 F00D   4640          BEQ  Y96
4942 AD8906 4650          LDA  LPTS
4945 38     4660          SEC
4946 E920   4670          SBC  #$20
4948 B002   4680          BCS  A91
494A A900   4690          LDA  #$00
494C 8D8906 4700 A91      STA  LPTS
494F E8     4710 Y96      INX
4950 E005   4720          CPX  #$05
4952 D0D5   4730          BNE  LOOP72
            4740 ;
            4750 ;now evaluate vulnerability to penetrations
            4760 ;
4954 A000   4770          LDY  #$00
4956 8C8B06 4780 LOOP54   STY  OCOLUM
4959 A200   4790          LDX  #$00
495B 8E8A06 4800 LOOP73   STX  COLUM
495E EC8B06 4810          CPX  OCOLUM
4961 F021   4820          BEQ  NXCLM
4963 BD8406 4830          LDA  LV,X
4966 38     4840          SEC
4967 F98406 4850          SBC  LV,Y
496A F018   4860          BEQ  NXCLM
496C 3016   4870          BMI  NXCLM
496E AA     4880          TAX
496F A901   4890          LDA  #$01
4971 0A     4900 LOOP74   ASL  A
4972 CA     4910          DEX
4973 D0FC   4920          BNE  LOOP74
4975 85C5   4930          STA  TEMPR
4977 AD8906 4940          LDA  LPTS
497A 38     4950          SEC
497B E5C5   4960          SBC  TEMPR
497D B002   4970          BCS  Y32
497F A900   4980          LDA  #$00
4981 8D8906 4990 Y32      STA  LPTS
4984 AE8A06 5000 NXCLM    LDX  COLUM
4987 E8     5010          INX
4988 E005   5020          CPX  #$05
498A D0CF   5030          BNE  LOOP73
498C C8     5040          INY
498D C005   5050          CPY  #$05
498F D0C5   5060          BNE  LOOP54
            5070 ;
            5080 ;now get overall line value weighted by danger vector
            5090 ;
4991 A6C2   5100          LDX  ARMY
4993 AC4806 5110          LDY  SECDIR
4996 D006   5120          BNE  Z18
4998 BD014D 5130          LDA  IFRN-55,X
499B 4CB549 5140          JMP  Z20
499E C001   5150 Z18      CPY  #$01
49A0 D006   5160          BNE  Z19
49A2 BD694D 5170          LDA  IFRE-55,X
49A5 4CB549 5180          JMP  Z20
49A8 C002   5190 Z19      CPY  #$02
```

```
49AA D006   5200         BNE  Z17
49AC BDD14D 5210         LDA  IFRS-55,X
49AF 4CB549 5220         JMP  Z20
49B2 BD394E 5230 Z17     LDA  IFRW-55,X
49B5 85C5   5240 Z20     STA  TEMPR
49B7 AE8906 5250         LDX  LPTS
49BA F013   5260         BEQ  Z49
49BC A5C7   5270         LDA  ACCLO
49BE 18     5280         CLC
49BF 65C5   5290 LOOP75  ADC  TEMPR
49C1 9009   5300         BCC  Y34
49C3 E6C8   5310         INC  ACCHI
49C5 18     5320         CLC
49C6 D004   5330         BNE  Y34
49C8 A9FF   5340         LDA  #$FF
49CA 85C8   5350         STA  ACCHI
49CC CA     5360 Y34     DEX
49CD D0F0   5370         BNE  LOOP75
            5380 ;
            5390 ;next secondary direction
            5400 ;
49CF C8     5410 Z49     INY
49D0 C004   5420         CPY  #$04
49D2 F01F   5430         BEQ  Y35
49D4 8C4806 5440         STY  SECDIR
            5450 ;
            5460 ;rotate array
            5470 ;
49D7 A218   5480         LDX  #$18
49D9 BD6306 5490 LOOP70  LDA  LINARR,X
49DC 9D4A06 5500         STA  BAKARR,X
49DF CA     5510         DEX
49E0 10F7   5520         BPL  LOOP70
49E2 A218   5530         LDX  #$18
49E4 BC787A 5540 LOOP71  LDY  ROTARR,X
49E7 BD4A06 5550         LDA  BAKARR,X
49EA 996306 5560         STA  LINARR,Y
49ED CA     5570         DEX
49EE 10F4   5580         BPL  LOOP71
49F0 4CBF48 5590         JMP  Y88
            5600 ;
            5610 ;
49F3 A5C8   5620 Y35     LDA  ACCHI
49F5 85CE   5630         STA  SQVAL
            5640 ;
            5650 ;get range to closest German into NBVAL
            5660 ;
49F7 A036   5670 Y65     LDY  #$36
49F9 A9FF   5680         LDA  #$FF
49FB 8D3A06 5690         STA  NBVAL
49FE B91B57 5700 LOOP59  LDA  ARRIVE,Y
4A01 C5C9   5710         CMP  TURN
```

```
4A03 F002   5720         BEQ  Z45
4A05 B021   5730         BCS  Y68
4A07 B90054 5740 Z45     LDA  CORPSX,Y
4A0A 38     5750         SEC
4A0B ED3406 5760         SBC  TARGX
4A0E 20304D 5770         JSR  INVERT
4A11 85C5   5780         STA  TEMPR
4A13 B99F54 5790         LDA  CORPSY,Y
4A16 38     5800         SEC
4A17 ED3506 5810         SBC  TARGY
4A1A 20304D 5820         JSR  INVERT
4A1D 18     5830         CLC
4A1E 65C5   5840         ADC  TEMPR
4A20 CD3A06 5850         CMP  NBVAL
4A23 B003   5860         BCS  Y68
4A25 8D3A06 5870         STA  NBVAL
4A28 88     5880 Y68     DEY
4A29 10D3   5890         BPL  LOOP59
            5900 ;
            5910 ;now determine whether to use offensive or defensive strategy
            5920 ;
4A2B A6C2   5930         LDX  ARMY
4A2D BD6106 5940         LDA  IFR-55,X
4A30 85C5   5950         STA  TEMPR
4A32 A90F   5960         LDA  #$0F
4A34 38     5970         SEC
4A35 E5C5   5980         SBC  TEMPR
4A37 900C   5990         BCC  A40
4A39 0A     6000         ASL  A          OK, let's fool the routine
4A3A 85C5   6010         STA  TEMPR
4A3C A909   6020         LDA  #$09
4A3E 38     6030         SEC
4A3F ED3A06 6040         SBC  NBVAL      I know that NBVAL<9 for all front line units
4A42 8D3A06 6050         STA  NBVAL
            6060 ;
            6070 ;now add NBVAL*IFR to SQVAL with defensive bonus
            6080 ;
4A45 AC3A06 6090 A40     LDY  NBVAL
4A48 D005   6100         BNE  Z24        this square occupied by a German?
4A4A 84CE   6110         STY  SQVAL      yes, do not enter!!!
4A4C 4CD64A 6120         JMP  EVALSQ
4A4F A4CD   6130 Z24     LDY  TRNTYP
4A51 B9B479 6140         LDA  DEFNC,Y
4A54 18     6150         CLC
4A55 6D3A06 6160         ADC  NBVAL
4A58 A8     6170         TAY
4A59 A900   6180         LDA  #$00
4A5B 18     6190         CLC
4A5C 65C5   6200 LOOP60  ADC  TEMPR
4A5E 9004   6210         BCC  Y69
4A60 A9FF   6220 Z22     LDA  #$FF
4A62 3003   6230         BMI  Y71
```

```
4A64 88      6240 Y69    DEY
4A65 D0F5    6250        BNE  LOOP60
             6260 ;
4A67 18      6270 Y71    CLC
4A68 65CE    6280        ADC  SQVAL
4A6A 9002    6290        BCC  X00
4A6C A9FF    6300        LDA  #$FF
4A6E 85CE    6310 X00    STA  SQVAL
             6320 ;
             6330 ;extract penalty if somebody else has dibs on this square
             6340 ;
4A70 A09E    6350        LDY  #$9E
4A72 B95A7A  6360 LOOP58 LDA  OBJX-55,Y
4A75 CD3406  6370        CMP  TARGX
4A78 D01E    6380        BNE  Y63
4A7A B96153  6390        LDA  OBJY-55,Y
4A7D CD3506  6400        CMP  TARGY
4A80 D016    6410        BNE  Y63
4A82 C4C2    6420        CPY  ARMY
4A84 F012    6430        BEQ  Y63
4A86 B91B57  6440        LDA  ARRIVE,Y
4A89 C5C9    6450        CMP  TURN
4A8B F002    6460        BEQ  Z44
4A8D B009    6470        BCS  Y63
4A8F A5CE    6480 Z44    LDA  SQVAL
4A91 E920    6490        SBC  #$20
4A93 85CE    6500        STA  SQVAL
4A95 4CD64A  6510        JMP  EVALSQ
4A98 88      6520 Y63    DEY
4A99 C037    6530        CPY  #$37
4A9B B0D5    6540        BCS  LOOP58
             6550 ;
             6560 ;now extract distance penalty
             6570 ;
4A9D BD0054  6580 Y60    LDA  CORPSX,X
4AA0 38      6590        SEC
4AA1 ED3406  6600        SBC  TARGX
4AA4 20304D  6610        JSR  INVERT
4AA7 85C5    6620        STA  TEMPR
4AA9 BD9F54  6630        LDA  CORPSY,X
4AAC 38      6640        SEC
4AAD ED3506  6650        SBC  TARGY
4AB0 20304D  6660        JSR  INVERT
4AB3 18      6670        CLC
4AB4 65C5    6680        ADC  TEMPR
4AB6 C907    6690        CMP  #$07
4AB8 9006    6700        BCC  Z48
4ABA A900    6710        LDA  #$00
4ABC 85CE    6720        STA  SQVAL     this square is too far away
4ABE F016    6730        BEQ  EVALSQ
             6740 ;
4AC0 AA      6750 Z48    TAX
```

```
4AC1 A901    6760        LDA  #$01
4AC3 0A      6770 LOOP77 ASL  A
4AC4 CA      6780        DEX
4AC5 10FC    6790        BPL  LOOP77
4AC7 85C5    6800        STA  TEMPR
4AC9 A5CE    6810        LDA  SQVAL
4ACB 38      6820        SEC
4ACC E5C5    6830        SBC  TEMPR
4ACE 85CE    6840        STA  SQVAL
4AD0 B004    6850        BCS  EVALSQ
4AD2 A900    6860        LDA  #$00
4AD4 85CE    6870        STA  SQVAL
             6880 ;
             6890 ;now evaluate this square
             6900 ;
4AD6 AC3306  6910 EVALSQ LDY  DIR
4AD9 A6C2    6920        LDX  ARMY
4ADB A5CE    6930        LDA  SQVAL
4ADD CD3106  6940        CMP  BVAL
4AE0 9006    6950        BCC  Y72
4AE2 8D3106  6960        STA  BVAL
4AE5 8C3206  6970        STY  BONE
4AE8 C8      6980 Y72    INY
4AE9 C004    6990        CPY  #$04
4AEB F006    7000        BEQ  Y73
4AED 8C3306  7010        STY  DIR
4AF0 4C1248  7020        JMP  DRLOOP
             7030 ;
4AF3 BD5A7A  7040 Y73    LDA  OBJX-55,X
4AF6 AC3206  7050        LDY  BONE
4AF9 3004    7060        BMI  Y74
4AFB 18      7070        CLC
4AFC 79F27B  7080        ADC  XINC,Y
4AFF 9D5A7A  7090 Y74    STA  OBJX-55,X
4B02 BD6153  7100        LDA  OBJY-55,X
4B05 AC3206  7110        LDY  BONE
4B08 3004    7120        BMI  Y75
4B0A 18      7130        CLC
4B0B 79F17B  7140        ADC  YINC,Y
4B0E 9D6153  7150 Y75    STA  OBJY-55,X
             7160 ;
             7170 ;
4B11 AD10D0  7180 TOGSCN LDA  TRIG0
4B14 F00C    7190        BEQ  A30       ignore game console if red button is down
4B16 A908    7200        LDA  #$08
4B18 8D1FD0  7210        STA  CONSOL
4B1B AD1FD0  7220        LDA  CONSOL
4B1E 2901    7230        AND  #$01
4B20 F00B    7240        BEQ  WRAPUP
4B22 CA      7250 A30    DEX
4B23 E037    7260        CPX  #$37
4B25 9003    7270        BCC  Y76
```

```
4B27 4C7347 7280         JMP  LOOP51
4B2A 4C7147 7290 Y76     JMP  MLOOP
            7300 ;
4B2D A29E   7310 WRAPUP  LDX  #$9E
4B2F 86C2   7320 LOOP62  STX  ARMY
4B31 BD1B57 7330         LDA  ARRIVE,X
4B34 C5C9   7340         CMP  TURN
4B36 9003   7350         BCC  Y78
4B38 4C1A4C 7360         JMP  Y77
4B3B BD5A7A 7370 Y78     LDA  OBJX-55,X
4B3E A003   7380         LDY  #$03
4B40 38     7390         SEC
4B41 FD0054 7400         SBC  CORPSX,X
4B44 1005   7410         BPL  Y79
4B46 A001   7420         LDY  #$01
4B48 20324D 7430         JSR  INVERT+2
4B4B 8C3D06 7440 Y79     STY  HDIR
4B4E 8D4106 7450         STA  HRNGE
4B51 A000   7460         LDY  #$00
4B53 BD6153 7470         LDA  OBJY-55,X
4B56 38     7480         SEC
4B57 FD9F54 7490         SBC  CORPSY,X
4B5A 1005   7500         BPL  Y80
4B5C A002   7510         LDY  #$02
4B5E 20324D 7520         JSR  INVERT+2
4B61 8C3E06 7530 Y80     STY  VDIR
4B64 8D4206 7540         STA  VRNGE
4B67 CD4106 7550         CMP  HRNGE
4B6A 9015   7560         BCC  Y81
4B6C 8D4306 7570         STA  LRNGE
4B6F AD4106 7580         LDA  HRNGE
4B72 8D4406 7590         STA  SRNGE
4B75 AD3D06 7600         LDA  HDIR
4B78 8D4006 7610         STA  SDIR
4B7B 8C3F06 7620         STY  LDIR
4B7E 4C934B 7630         JMP  Y82
4B81 8D4406 7640 Y81     STA  SRNGE
4B84 8C4006 7650         STY  SDIR
4B87 AD4106 7660         LDA  HRNGE
4B8A 8D4306 7670         STA  LRNGE
4B8D AC3D06 7680         LDY  HDIR
4B90 8C3F06 7690         STY  LDIR
4B93 A900   7700 Y82     LDA  #$00
4B95 8D4706 7710         STA  RCNT
4B98 8D3B06 7720         STA  RORD1
4B9B 8D3C06 7730         STA  RORD2
4B9E AD4306 7740         LDA  LRNGE
4BA1 18     7750         CLC
4BA2 6D4406 7760         ADC  SRNGE
4BA5 8D4606 7770         STA  RANGE
4BA8 F05C   7780         BEQ  Y86
4BAA AD4306 7790         LDA  LRNGE
4BAD 4A     7800         LSR  A
4BAE 8D4506 7810         STA  CHRIS
            7820 ;
4BB1 AD4506 7830 LOOP61  LDA  CHRIS
4BB4 18     7840         CLC
4BB5 6D4406 7850         ADC  SRNGE
4BB8 8D4506 7860         STA  CHRIS
4BBB 38     7870         SEC
4BBC ED4606 7880         SBC  RANGE
4BBF B005   7890         BCS  OVRFLO
4BC1 AD3F06 7900         LDA  LDIR   FBC4 9006   7910         BCC  STIP
4BC6 8D4506 7920 OVRFLO  STA  CHRIS
4BC9 AD4006 7930         LDA  SDIR
4BCC 8D3306 7940 STIP    STA  DIR
4BCF AD4706 7950         LDA  RCNT
4BD2 2903   7960         AND  #$03
4BD4 A8     7970         TAY
4BD5 85C5   7980         STA  TEMPR
4BD7 AD4706 7990         LDA  RCNT
4BDA 4A     8000         LSR  A
4BDB 4A     8010         LSR  A
4BDC AA     8020         TAX
4BDD AD3306 8030         LDA  DIR
4BE0 88     8040 Y85     DEY
4BE1 3005   8050         BMI  Y84
4BE3 0A     8060         ASL  A
4BE4 0A     8070         ASL  A
4BE5 4CE04B 8080         JMP  Y85
            8090 ;
4BE8 A4C5   8100 Y84     LDY  TEMPR
4BEA 5D3B06 8110         EOR  RORD1,X
4BED 39DE5F 8120         AND  MASKO,Y
4BF0 5D3B06 8130         EOR  RORD1,X
4BF3 9D3B06 8140         STA  RORD1,X
4BF6 AE4706 8150         LDX  RCNT
4BF9 E8     8160         INX
4BFA 8E4706 8170         STX  RCNT
4BFD E008   8180         CPX  #$08
4BFF B005   8190         BCS  Y86
4C01 EC4606 8200         CPX  RANGE
4C04 90AB   8210         BCC  LOOP61
4C06 A6C2   8220 Y86     LDX  ARMY
4C08 AD3B06 8230         LDA  RORD1
4C0B 9D145E 8240         STA  WHORDS,X
4C0E AD3C06 8250         LDA  RORD2
4C11 9DB35E 8260         STA  WHORDH,X
4C14 AD4706 8270         LDA  RCNT
4C17 9D755D 8280         STA  HMORDS,X
            8290 ;
4C1A CA     8300 Y77     DEX
4C1B E037   8310         CPX  #$37
```

```
4C1D 9003   8320        BCC  Y87
4C1F 4C2F4B 8330        JMP  LOOP62
4C22 60     8340 Y87    RTS
            8350 ;
            8360 ;Subroutine CALIFR determines individual force ratios
            8370 ;in all four directions
            8380 ;
4C23 A000   8390 CALIFR LDY  #$00       initialize vectors
4C25 8C7C06 8400        STY  IFR0
4C28 8C7D06 8410        STY  IFR1
4C2B 8C7E06 8420        STY  IFR2
4C2E 8C7F06 8430        STY  IFR3
4C31 8C8C06 8440        STY  IFRHI
4C34 C8     8450        INY
4C35 84CC   8460        STY  RFR
4C37 BD0054 8470        LDA  CORPSX,X
4C3A 8D8006 8480        STA  XLOC
4C3D BD9F54 8490        LDA  CORPSY,X
4C40 8D8106 8500        STA  YLOC
4C43 A09E   8510        LDY  #$9E
4C45 B91B57 8520 LOOP53 LDA  ARRIVE,Y
4C48 C5C9   8530        CMP  TURN
4C4A B021   8540        BCS  Z07
4C4C B99F54 8550        LDA  CORPSY,Y
4C4F 38     8560        SEC
4C50 ED8106 8570        SBC  YLOC
4C53 8D8306 8580        STA  TEMPY      save signed vector
4C56 20304D 8590        JSR  INVERT
4C59 85C5   8600        STA  TEMPR
4C5B B90054 8610        LDA  CORPSX,Y
4C5E 38     8620        SEC
4C5F ED8006 8630        SBC  XLOC
4C62 8D8206 8640        STA  TEMPX
4C65 20304D 8650        JSR  INVERT
4C68 18     8660        CLC
4C69 65C5   8670        ADC  TEMPR
4C6B C909   8680 Z21    CMP  #$09       no point in checking if he's too far
4C6D B067   8690 Z07    BCS  Y48
4C6F 4A     8700        LSR  A
4C70 85C5   8710        STA  TEMPR      this is half of range to unit
            8720 ;
            8730 ;now select which IFR gets this German
            8740 ;
4C72 AD8206 8750        LDA  TEMPX
4C75 1010   8760        BPL  Z00
4C77 AD8306 8770        LDA  TEMPY
4C7A 1029   8780        BPL  Z01
4C7C A202   8790        LDX  #$02
4C7E CD8206 8800        CMP  TEMPX
4C81 B031   8810        BCS  Z02
4C83 A201   8820        LDX  #$01
4C85 902D   8830        BCC  Z02
4C87 AD8306 8840 Z00    LDA  TEMPY
4C8A 100E   8850        BPL  Z03
4C8C 20324D 8860        JSR  INVERT+2
4C8F A202   8870        LDX  #$02
4C91 CD8206 8880        CMP  TEMPX
4C94 B01E   8890        BCS  Z02
4C96 A203   8900        LDX  #$03
4C98 901A   8910        BCC  Z02
4C9A A200   8920 Z03    LDX  #$00
4C9C CD8206 8930        CMP  TEMPX
4C9F B013   8940        BCS  Z02
4CA1 A203   8950        LDX  #$03
4CA3 900F   8960        BCC  Z02
4CA5 AD8206 8970 Z01    LDA  TEMPX
4CA8 20324D 8980        JSR  INVERT+2
4CAB A201   8990        LDX  #$01
4CAD CD8306 9000        CMP  TEMPY
4CB0 B002   9010        BCS  Z02
4CB2 A200   9020        LDX  #$00
4CB4 B9DD55 9030 Z02    LDA  CSTRNG,Y
4CB7 4A     9040        LSR  A
4CB8 4A     9050        LSR  A
4CB9 4A     9060        LSR  A
4CBA 4A     9070        LSR  A
4CBB C037   9080 Z11    CPY  #$37
4CBD 900C   9090        BCC  Z12
4CBF 18     9100        CLC
4CC0 65CC   9110        ADC  RFR
4CC2 9002   9120        BCC  Z13
4CC4 A9FF   9130        LDA  #$FF
4CC6 85CC   9140 Z13    STA  RFR
4CC8 4CD64C 9150        JMP  Y48
4CCB 18     9160 Z12    CLC
4CCC 7D7C06 9170        ADC  IFR0,X
4CCF 9002   9180        BCC  Z05
4CD1 A9FF   9190        LDA  #$FF
4CD3 9D7C06 9200 Z05    STA  IFR0,X
4CD6 88     9210 Y48    DEY
4CD7 F003   9220        BEQ  Z06
4CD9 4C454C 9230        JMP  LOOP53
            9240 ;
4CDC A203   9250 Z06    LDX  #$03
4CDE A900   9260        LDA  #$00
4CE0 18     9270 Y37    CLC
4CE1 7D7C06 9280        ADC  IFR0,X
4CE4 9002   9290        BCC  Y36
4CE6 A9FF   9300        LDA  #$FF
4CE8 CA     9310 Y36    DEX
4CE9 10F5   9320        BPL  Y37
            9330 ;
            9340 ;
4CEB 0A     9350        ASL  A
```

```
4CEC 2E8C06 9360         ROL  IFRHI
4CEF 0A     9370         ASL  A
4CF0 2E8C06 9380         ROL  IFRHI
4CF3 0A     9390         ASL  A
4CF4 2E8C06 9400         ROL  IFRHI
4CF7 0A     9410         ASL  A
4CF8 2E8C06 9420         ROL  IFRHI
4CFB A200   9430         LDX  #$00
4CFD 38     9440         SEC
4CFE E5CC   9450 Z16     SBC  RFR
4D00 B006   9460         BCS  Z14
4D02 CE8C06 9470         DEC  IFRHI
4D05 38     9480         SEC
4D06 3004   9490         BMI  Z15
4D08 E8     9500 Z14     INX
4D09 4CFE4C 9510         JMP  Z16
4D0C 8A     9520 Z15     TXA
4D0D A6C2   9530         LDX  ARMY
4D0F 18     9540         CLC
4D10 6D9206 9550         ADC  OFR        remember strategic situation
4D13 6A     9560         ROR  A          average strategic with tactical
4D14 9D6106 9570         STA  IFR-55,X
            9580 ;
            9590 ;keep a record of danger vector
            9600 ;
4D17 AD7C06 9610         LDA  IFR0
4D1A 9D014D 9620         STA  IFRN-55,X
4D1D AD7D06 9630         LDA  IFR1
4D20 9D694D 9640         STA  IFRE-55,X
4D23 AD7E06 9650         LDA  IFR2
4D26 9DD14D 9660         STA  IFRS-55,X
4D29 AD7F06 9670         LDA  IFR3
4D2C 9D394E 9680         STA  IFRW-55,X
4D2F 60     9690         RTS
            9700 ;
4D30 1005   9710 INVERT  BPL  Z46
4D32 49FF   9720         EOR  #$FF
4D34 18     9730         CLC
4D35 6901   9740         ADC  #$01
4D37 60     9750 Z46     RTS
            9760 ;
4D38        9770 IFRN    *=   *+104
4DA0        9780 IFRE    *=   *+104
4E08        9790 IFRS    *=   *+104
4E70        9800 IFRW    *=   *+104
4ED8        9810         .END
```

DISCLAIMER OF WARRANTY AND LIABILITY ON COMPUTER PROGRAMS

Neither Atari, Inc. ("ATARI"), nor its software supplier, distributor, or dealers make any express or implied warranty of any kind with respect to this computer software program and/or material, including, but not limited to warranties of merchantability and fitness for a particular purpose. This computer program software and/or material is distributed solely on an "as is" basis. The entire risk as to the quality and performance of such programs is with the purchaser. Purchaser accepts and uses this computer program software and/or material upon his/her own inspection of the computer software program and/or material, without reliance upon any representation or description concerning the computer program software and/or material. Should the computer program software and/or material prove defective, purchaser and not ATARI, its software supplier, distributor, or dealer, assumes the entire cost of all necessary servicing, repair, or correction, and any incidental damages.

In no event shall ATARI, or its software supplier, distributor, or dealer be liable or responsible to a purchaser, customer, or any other person or entity with respect to any liability, loss, incidental or consequential damage caused or alleged to be caused, directly or indirectly, by the computer program software and/or material, whether defective or otherwise, even if they have been advised of the possibility of such liability, loss, or damage.

LIMITED WARRANTIES ON MEDIA AND HARDWARE ACCESSORIES

ATARI warrants to the original consumer purchaser that the media on which the computer software program and/or material is recorded, including computer program cassettes or diskettes, and all hardware accessories are free from defects in materials or workmanship for a period of 30 days from the date of purchase. If a defect covered by this limited warranty is discovered during this 30-day warranty period, ATARI will repair or replace the media or hardware accessories, at ATARI's option, provided the media or hardware accessories and proof of date of purchase are delivered or mailed, postage prepaid, to the ATARI Program Exchange.

This warranty shall not apply if the media or hardware accessories (1) have been misused or show signs of excessive wear, (2) have been damaged by playback equipment or by being used with any products not supplied by ATARI, or (3) if the purchaser causes or permits the media or hardware accessories to be serviced or modified by anyone other than an authorized ATARI Service Center. Any applicable implied warranties on media or hardware accessories, including warranties of merchantability and fitness, are hereby limited to 30 days from the date of purchase. Consequential or incidental damages resulting from a breach of any applicable express or implied warranties on media or hardware accessories are hereby excluded. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. Some states also do not allow the exclusion or limitation of incidental or consequential damage, so the above limitation or exclusion may not apply to you.

ATARI PROGRAM EXCHANGE

REVIEW FORM

We're interested in your experiences with APX programs and documentation, both favorable and unfavorable. Many software authors are willing and eager to improve their programs if they know what users want. And, of course, we want to know about any bugs that slipped by us, so that the software author can fix them. We also want to know whether our documentation is meeting your needs. You are our best source for suggesting improvements! Please help us by taking a moment to fill in this review sheet. Fold the sheet in thirds and seal it so that the address on the bottom of the back becomes the envelope front. Thank you for helping us!

1. Name and APX number of program _____

2. If you have problems using the program, please describe them here.

_____

_____

_____

3. What do you especially like about this program?

_____

_____

_____

4. What do you think the program's weaknesses are?

_____

_____

_____

5. How can the catalog description be more accurate and/or comprehensive?

_____

_____

6. On a scale of 1 to 10, 1 being "poor" and 10 being "excellent", please rate the following aspects of this program?

_____ Easy to use
_____ User-oriented (e.g., menus, prompts, clear language)
_____ Enjoyable
_____ Self-instructive
_____ Useful (non-game software)
_____ Imaginative graphics and sound

7. Describe any technical errors you found in the user instructions (please give page numbers).

8. What did you especially like about the user instructions?

9. What revisions or additions would improve these instructions?

10. On a scale of 1 to 10, 1 representing "poor" and 10 representing "excellent", how would you rate the user instructions and why?

11. Other comments about the software or user instructions:

STAMP

ATARI Program Exchange
P.O. Box 427
155 Moffett Park Drive, B-1
Sunnyvale, CA 94086

[seal here]