NAME

xbattle – a multi-user battle strategy game

SYNOPSIS

xbattle [*–color display...*] [*–option* [*option_ar g ...*] ...]

DESCRIPTION

A board-based, multi-player, real-time battle strategy game.

Assign each player a team color and display, and any number of game options where required. Valid <*color*> options can be either a monochrome tone, **–black –white –dark –light**, or a color, **–red –green –blue**. is the name of the X display for each player (the display number and screen number are optional). Command line arguments can be supplied in any order. For a quick introduction, go straight to the EXAMPLES section below. Also, see the tutorials and example scripts which are supplied with the game.

-<c1>	<display>	assign color to display
-<c1>_<c2>	<display>	assign colors to display
-ai	<ply> <alg> <sk>	make player <ply> an AI player using algo <alg> with skill <sk>
-area		troop strength proportional to cell area
-attack		allow use of attack key
-armies	<int>	number of initial ordered armies
-basemap		use map scheme, bases visible
-bases	<int>	number of initial ordered bases
-board	<int>	size of board (in cells, width=height)
-boardx	<int>	width of board (in cells)
-boardy	<int>	height of board (in cells)
-border	<int>	border width around board
-bound		allow drag-bounding direction sets
-build	<int>	build cities, <int> segments to complete
-build_cost	<int>	cost to build city segment
-build_limit	<int>	limit cities each side can build
-cell	<int>	diameter of cell (in pixels)
-color	<str> <r><g><b>	set RGB values for color <str>
-color_inverse	<str> <str2>	set color <str> inverse to <str2>
-decay	<int>	make troops slowly die off
-diamond		use diamond board tiles
-dig	[int]	allow terrain lowering, [int] steps
-dig_cost	<int>	cost of each dig step
-digin	<int>	provide entrenchment
-disrupt		allow enemies to break supply lines
-draw	<int>	specify a troop drawing method
-dump	<file>	dump configuration to <file>
-edit	[file]	interactively edit xbattle board
-erode	<int>	make unused paths erode
-erode_thresh	<int>	threshold for erosion
-farms	<int>	troops slowly grow
-fight	<int>	intensity of fighting
-fill	[int]	allow terrain raising, [int] steps
-fill_cost	<int>	cost of each fill step
-forest	<int>	density of forest
-forest_color	<int> <r><g><b>	RGB values for forest level <int>
-forest_tones	<int>	number of allowable forest levels
-grid		show board grid
-guns	<int>	range of artillery
-guns_cost	<int>	cost of each artillery shell

-guns_damage	<int>	damage done by each artillery shell
-help		print usage information and exit
-hex		use hexagonal board tiles
-hidden		do not show enemy direction vectors
-hills	<int>	slope of hills
-hill_color	<int> <r><g><b>	RGB values for hill level <int>
-hill_tones	<int>	number of allowable hill levels
-horizon	[int]	can’t see enemy past [int] cells
-is_server	<port>	run server listening on given port for connections (player-specific option)
-load	[file]	load board from [file] (default is xbattle.xbt)
-localmap		use map scheme, disappearing terrain
-manage		allow managed control of operations
-manpos		enable manual positioning of board
-map		use basic map scheme
-march	<int>	number of delays between marches
-maxval	<int>	maximum cell troop capacity
-militia	<int>	density of randomly distributed troops
-move	<int>	speed of troop flow
-new_colormap		use a new (private) colormap, if possible
-nospigot	[int]	cease attack if enemy/you ratio > [int]
-octagon		use octagonal/square board tiles
-options	<file>	read xbattle options from <file>
-opt_file.xbo		shorthand for -options <file>.xbo
-overwrite		only use terrain from load file
-para	<int>	range of paratroopers
-para_cost	<int>	cost of each paratrooper
-para_damage	<int>	invading strength of each paratrooper
-peaks	<int>	number of terrain peaks (and troughs)
-peak_bias	<float>	peak distribution bias (0.0-2.0)
-rbases	<int>	number of initial randomly-distributed bases (in cells)
-rbase_range	<int>	minimum distance of rbase from enemy base
-repeat		allow repeat of last mouse command
-replay	[file]	replay stored game from [file]
-reserve		allow reserve of troops
-scuttle	[int]	enable city scuttling, [int] segments
-scuttle_cost	<int>	cost of scuttle
-sea	<int>	pervasiveness (and levels) of sea
-sea_block		use area-fills rather than hue-fills
-sea_color	<int> <r><g><b>	RGB values for sea level <int>
-sea_tones	<int>	number of allowable sea levels
-sea_value	<float>	darkness of seas for b/w games
-seed	<int>	random number generator seed
-speed	<int>	speed of updates
-square		use square board tiles
-stipple	<str> 8*<hex>	set stipple (b/w) pattern for color <str>
-store	[file]	store game in [file] for later replay
-towns	<int>	density of randomly-distributed towns
-triangle		use triangular board tiles
-trough_bias	<float>	trough distribution bias (0.0-2.0)
-use_server	<host> <port> <ply>	connect to given server as player number <ply>
-visual	<id>	run on specified visual
-win_trad		victory after killing all other forces
-win_army	<float>	victory if over a percentage of all armies
-win_land	<float>	victory if over a percentage of land

-win_timeout	<int>	victory after a certain time
-win_pos	<x> <y>	victory if player occupies position
-win_wait	<int>	wait period after victory
-wrap		allow wrapping around edges of board
-xpos	<int>	x position of board on display
-ypos	<int>	y position of board on display

CONTROLS

LFT MOUSE:	toggle command vector
MID MOUSE:	clear and set new command vector
RGT MOUSE:	repeat previous command (-repeat)
SHIFT-LFT MOUSE:	march (-march) fork move (else)
SHIFT-MID MOUSE:	force march (-march) fork move (else)
SHIFT-RGT MOUSE:	paratroops (-para)
CTRL-RGT MOUSE:	artillery (-guns)
'a':	attack enemy square (-attack)
'b':	build base (-build)
'B':	build full base (-build and -manage)
's':	scuttle base (-scuttle)
'f':	fill terrain (-fill)
'F':	fill full terrain (-fill and -manage)
'd':	dig terrain (-dig)
'D':	dig full terrain (-dig and -manage)
'p':	paratroops (-para)
'P':	paratroops - on (-para and -manage)
'g':	artillery (-guns)
'G':	artillery - on (-guns and -manage)
'z':	cancel all movement
'c':	cancel managed operation (-manage)
'0'-'9':	reserve (-reserve)

COMMANDS IN GAMEBOARD

RETURN:	enter text message
CRTL-'s':	pause game
CRTL-'q':	resume game
CRTL-'p':	save game state to map file

COMMANDS IN TEXT AREA

CONTROL-c:	quit the game
CONTROL-w:	quit game but watch others play on
CONTROL-g:	ring bell on all game displays
CONTROL-p:	dump the current game state
OTHER CHARACTER:	append to message string

OVERVIEW

xbattle is a concurrent multi-player battle strategy game that captures the dynamics of a wide range of military situations. The game board is a matrix of game cells (such as squares or hexagons) which can be occupied by troops of various sides represented by colors or shades. The number of troops in a cell is indicated by the size of a colored troop circle (or square) within that cell. The troops are commanded by clicking the mouse near the edge of the cell in the direction that the movement is to take place. The command will be acknowledged by the appearance of a movement vector, and thereafter, in each update cycle, a certain proportion of the troops will move from the source cell to the destination cell until the source cell is exhausted. Movement vectors can be set in several directions at once, in which case the movement is divided evenly between the vector directions, and the command remains active until canceled. Thus a trail of cells can be set up as a supply line that will deliver troops steadily at its endpoint. The movement vector remains active even if the number of troops in that cell is zero, although the movement vector will then be displayed at half length. The game is concurrent instead of turn-based, meaning that commands may be given continuously by all players without waiting for the another player.

TEAM SIDES AND PLAYER OPTIONS

−<*color*>, **−color**, **−color_inverse**, **−stipple**

The game is started from one display, and each player must play from a separate display, players being assigned to a color team by the command line option **−**<*color*> <*display*>. The parameter<*color*> determines the color of the troops of that team, which can be either a monochrome tone like black, white, dark, light, or a true color like red, green, blue, although the true colors will appear on a monochrome monitor as either black or white with an identifying character in each troop marker which is the first letter of the color name. For instance, the team color **−red** would appear on a monochrome monitor as black with a letter "R" in the middle of each troop marker. The legal team color names can be selected from any color defined in the file */usr/lib/X11/rgb.txt* which includes such bizarre entries as "LavenderBlush", "MistyRose", "PapayaWhip" as well as the standard "red", "green", "blue" and "black" and "white" etc. Alternatively, colors can be defined individually in the default file (*~/.xbattle*), an option file (see OPTION FILE OPTIONS section below), or in the command line itself using the **−color** <*str*> <*r*> <*g*> <*b*> option. With this option, the color is given by<*str*>, and the red green and blue components by<*r*>, <*g*>, and<*b*> respecti vely, in the range 0 - 255. A black and white pattern can be assigned to correspond to color name<*str*> via the **−stipple** <*str*> 8 x <*hex*> *option, where the binary* breakdown of each of eight hex numbers (in C form like "0xa4") specifies one of the eight rows of the pattern.

By default, **xbattle** supports the colors "dark", "light", "gator", "brick", which appear as bitmap textures on monochrome monitors, allowing monochrome players to have six distinguishable team colors. A number of people can be assigned to the same team by repeating the color for different displays, for example "-red display1 -red display2", and each member of the team will be able to command any troops of that team. The <*display*> argument designates the name of the display on which the team of that color is playing, so each player must be given a color and a display. Display names can be found with the unix command "who", which will list display names for users in the last column like (cnsxk:0.0). The system console is always designated unix:0.0. The display name can be modified for remote games, for example the terminal cnsxk:0.0 on park.bu.edu (email address of machine "park") is designated cnsxk.bu.edu:0.0 . XBattle recognizes :0.0 as the default (screen zero on the display), so the :0.0 may be omitted from the display name. XBattle also recognizes a special display name "me", which means the display from which the program is started. When playing between color and monochrome displays the colors can be specified more exactly by concatenating a color name with a monochrome name, for example "-red_white" (color first), which would display that team as red on color monitors and white on monochrome monitors. All command line flags and arguments for the game can be given in any order as long as the argument directly follows its flag, and most arguments are scaled to range from 1 to 10, with 5 being the default value. It is also possible to set different game parameters for the different displays, so that the game can be biased to favor a less experienced player (see BIASED GAMES below).

AI PLAYER OPTIONS

−ai

Computer-controlled players can be added to a game in place of human players. Multiple computer players may be used in the same game. A single side can have any mix of computer and human players, though the computer players are difficult to work with.

The **-ai** <*player*> <*algo*> <*skill*> option will make player number <*player*> be controlled by AI algorithm number <*algo*>. The player number may range from 0 to N-1 where N is the total number of players specified on the command line. The numbers are assigned in the same order as the command line. For example, in the game...

```
  xbattle -red dispa:0 -green dispb:0 -blue dispc:0
```

red is player 0, green is player 1, and blue is player 2.

The AI number may currently be 1 (for the original algorithm) or 2 (for the new AI algorithm). The <*skill*> agument determines the skill level or other characteristics of the computer player. The actual meaning is specific to the AI algorithm.

Algorithm 1
: The skill level mainly affects how many moves per timeunit the AI is allowed to make - a move is here an added or removed vector from a cell. Setting the skill level to 0.5 - 1.0 should be enough to provide a good challenge to novice players. Higher skill levels mean more moves and hence, a better opponent. The **-speed** option multiplies the effect of this setting.

Algorithm 2
: Skill levels for this algorithm range from 1 to 10. Each level defines a particular fighting style. Generally, the higher the level the better, but some are really special. The following is the complete list of skill levels available.

1 Pacifist	Average development rate, rarely attacks
2 Trainee	Slow development, average attack skill, good organisation
3 Commander	Slow development, good attack skill, good organisation
4 Elite	Good development, good attack skill, good organisation
5 Quick	Fast expansion, average attack
6 Psyco	Average development, frantic attacks, poor organisation
8 Barbarian	Fast development, fast attack, poor organisation
10 Ultimate	The ultimate fighter (well, the best this algorithm can do)
other	Any other skill value will result in default settings

OPTION FILE OPTIONS

−options

A large number of command line options are available to define the parameters of the game. In essence, **xbattle** is many thousands of games rolled into one. The options can be presented in any order, and may be typed in with the command line, or they can be stored in an options file (default filename is default.xbo), or some can be stored and others added to the command line. The format for the options file is exactly the same as the format for the command line except that in the file each option (plus argument, where applicable) is placed on a separate line. So, for example, the game...

```
  xbattle -black me -white cnsxk:0.0 -armies 4 -farms 5 -attack
```

could also be played with the command...

```
  xbattle -black me -white cnsxk:0.0 -options myoptions.xbo
```

or alternatively with the shorthand version...

```
  xbattle -black me -white cnsxk:0.0 -myoptions.xbo
```

where the file myoptions.xbo consists of the lines...

```
  -armies 4
  -farms 5
  -attack
```

If the specified options file cannot be found in the current directory, **xbattle** will search the default xbo directory DEFAULT_XBO_DIR, which can be specified at compile time in the makefile.

XBattle checks for the presence of the default file*~/.xbattle*, in which options can be set in adv ance, saving the trouble of having to set them at run time or include an options files. The default file is typically used to set up commands which are used in nearly every game at a particular site. Thus a typical default file might contain color (and black and white) definitions, cell size, artillery damage, and the like. Option files, on the other hand, are typically used to define specific scenarios involving unique combinations of command line arguments. Conflicting commands in the default and options file are resolved in favor of the options file.

INITIAL TROOP OPTIONS

−bases, **−rbases**, **−rbase_range**, **−armies**, **−militia**

Initial troop allocation is controlled by several command options, including**-bases** <*n*>, **-rbases** <*n*>, **-armies** <*n*>, and **-militia** <*n*>. Armies and militia are troops on the gameboard, whereas bases, which are indicated by circles on the gameboard, provide a steady supply of troops. The **-bases** <*n*> option allocates <*n*> bases to each team, symmetrically arranged on the game board, whereas **-rbases** arranges them randomly (which works well with the **-horizon** option). The minimum distance between enemy bases (in cells) can optionally be set using the **-rbase_range** <*n*> command. Note that large values of <*n*> may not allow any valid rbase allocation, in which case xbattle will exit with an error message. The **-armies** option allocates <*n*> armies (full troop cells) symmetrically arrayed, whereas **-militia** scatters militia of random strengths to random locations, with a probabilistic density of <*n*>. At least one of these four options is required to provide initial troops for the game, and they may be used in arbitrary combinations.

RESUPPLY OPTIONS

−towns, **−farms**, **−decay**, **−erode**, **−erode_thresh**

The bases created by the **−bases** or **−rbases** produce a steady supply of fresh troops. The bases can be occupied by an opposing team, with the troops produced by such bases are always the color of the occupying force. The capture of all bases thus becomes the strategic objective of the game. This arrangement simulates desert warfare, as long and tenuous supply lines develop between the base and the battle areas. Another form of resupply is provided by the command option "-towns <n>". This produces a number of smaller unoccupied supply sources scattered randomly over the game board at a density determined by the argument <*n*>, and with random rates of troop production, indicated by the radius of the circle on the game board. Towns must be occupied by a team to begin producing troops. This option simulates yet a larger scale of operation as the combatants battle to occupy the towns. A more distributed form of resupply is evoked by the command option "-farms <n>" whereby every cell of the game board will produce troops as soon as it is occupied, at a rate proportional to the argument <*n*>, and the strategic objective becomes the occupation of the largest areas of the gameboard. This option simulates a yet larger scale of operation and requires complex management of resources to concentrate the distributed resources and deliver them to the battle front. In large scale scenarios additional realism may be added by using the "-decay <n>" option whereby the troop strength in all troop cells decays constantly in proportion to the value of the decay argument. This reflects the fact that armies constantly consume resources even while they are idle, and an army without constant resupply will wither away. With the decay option, a set of bases, towns or farms can only support armies of limited size, and the armies will dynamically grow or shrink until they reach that size. Since this number includes the troops that make up the supply line, the fighting power of an army diminishes with the length of the supply line. The default decay value is zero, i.e. no decay. All the resupply options can be used in any combination. The "-erode <n>" command doesn't affect resuply, per se, but it does effect the movement vectors through which troops flow by causing them to erode away as they grow older. All movement vectors in a cell will be unset at a random time not to be less than <*n*> update cycles, with probability of erosion for each subsequent cycle determined by the "-erode_thresh <m>" argument, where <*m*> is the percentage chance of erosion.

ENHANCED MOVEMENT COMMAND OPTIONS

−repeat, **−bound**, **−attack**, **−march**, **−reserve**

With the option **-repeat** you can repeat the last command using the right mouse. If for example your last command to a cell consisted of a "move up" command by clicking near the top edge of the cell, you can now command other cells to also move up by clicking in those cells with the right mouse. That way you no longer have to aim your click exactly at the top side of those cells, but can click the right mouse anywhere in that cell, which saves time. This command is supported in biased games - i.e. it can be set for one team but not another. Commands can be made to apply to more than one cell with the option**-bound**. This is achieved by defining a bounding rectangle within which the command is valid. For instance, to command a block of cells to all move up simultaneously, you place your mouse near the top edge of a cell (may be unoccupied, or enemy occupied) and press the button (setting the command "go up", then you drag the mouse to another game cell where you release the button. The start and end cells of the mouse drag define the opposite corners of a rectangle within which all the game cells occupied by your troops receive the command "go up". The **-attack** option makes quick, multiple front attacks possible. By issuing an "a" command in an enemy cell, all adjacent friendly troops will automatically alter their movement vectors so as to attack the enemy cell, and only that cell. The **-reserve** option allows a player to define a level of reserves to remain in the cell despite any movement vectors. For instance a reserve level of 5 would ensure that the cell will maintain a reserve of 50% capacity, and movement out of that cell will only occur with troops in excess of the reserve level. This is extremely useful when a supply line must pass through a strategically important part of the board. The reserve level is set in a particular game cell by pointing to that cell with the mouse and striking a number key, "1" for 10% reserves, "2" for 20% reserves, and so forth up to "9" for 90% reserves.

With the option "-march <n>", troops may be commanded to march in a particular direction and to continue in that direction without further commands. March commands are activated with shift left or shift middle mouse button. For example, if you click near the top edge of a cell with "shift left mouse", the troops will begin to march up, and on arrival in the next cell they will transfer the march command to that cell so that they will continue marching upwards to the next cell, and so forth. If a marching column encounters hostile forces the march command is canceled and the column stops. To prevent marching columns from traveling much faster than manually commanded troops, the march argument <*n*> defines the number of game update cycles that the troops must wait in each new cell before marching on to the next cell, so that "-march 1" will result in a fast march, whereas "-march 10" will be slow. The "march command" is indicated on the game board by a double command vector (looks like an "=" sign) in the appropriate direction, and the march command is always passed on to the head of the column. March commands may be set in cells that are NOT occupied by your troops, and will be activated when a marching column arrives in that cell. This allows you to define turns in the path of the marching column to avoid obstacles. A "stop march" command may also be set to program the marching column to stop in that cell. This is achieved by clicking "shift left mouse" in the center of that cell, and will be displayed as an empty box in that cell. When set with the left mouse, the march vector is overwritten on to existing command vectors encountered in the march path, whereas when set with the middle mouse the march vector removes and replaces existing command vectors. March commands are canceled by clicking on the cell without the shift key. March commands may be set in cells that are beyond the visible horizon in the normal way , and will appear as a double vector in that cell so long as that cell is not a "sea" cell. If the target cell contains invisible enemy troops, then the march command vectors will appear initially, but disappear again as soon as the enemy is approached close enough to be visible. March commands are specific to the team that sets them, and different march commands may be set by different teams in the same game cell. The double command vectors are visible only to the team that sets them.

GAME PLAY

–fight, **–speed**, **–move**, **–seed**, **–digin**, **–nospigot**, **–disrupt**, **–maxval**

Whenever troops of different colors occupy the same game cell, a battle ensues, indicated by concentric markers of the two colors, and a "crossed swords" (X) symbol. During battle, one or both sides can incur losses according to a random nonlinear function that disproportionately favors the more numerous troops. The steepness of the nonlinearity, i.e. the advantage given to the more numerous side, is controlled by the **–fight** parameter. A small value will produce lengthy drawn out battles which favor a defensive strategy, whereas a large value produces quick decisive battles where the random element is more significant, favoring an offensive strategy even against superior odds. In the absence of the **–fight** option, the default value of 5 is used. The **–fight** parameter is also automatically modulated by the game speed parameter (**-speed**) in order to slow down battles in fast games and vice versa. Since only 1/3 of the troops can enter a cell in each update cycle (with the default **–move** 5), attackers of a full cell are always outnumbered initially, unless a coordinated attack is launched from three sides simultaneously. The**–mo ve** argument thus has a significant influence on the efficacy of an attack. The **–disrupt** option dictates that when a game cell comes under attack, all its command vectors are immediately canceled, breaking supply lines which must be repaired by hand after the attack. In other words, there can be no movement under fire, and even small forces can be used to provide covering fire to "pin down" a larger force, at least until they are counter-attacked and eliminated. A side effect of this option is that when an attacking cell is counterattacked, both cells attempt to cancel each other's movement, i.e. to interrupt the attack. The cell that is updated next will prevail, canceling the command vector of the other cell. Since the game cells are updated in a random sequence, there is no telling which cell will prevail, and the commander must click repeatedly to renew the command vector in order to press home the attack under opposition. This simulates the tactical situation where a commander must personally intervene to ensure the maximal effort is applied at the most critical points of the battle. The **–seed** <*n*> option simply sets the seed of the random number generator to <*n*>, which is useful for recreating scenarios. By default the random number generator is seeded with a combination of the system time and process ID number --- a more-or-less random number.

In each update cycle some fraction of the troops in a game cell move to adjacent cells indicated by the command vectors. The default fraction is 1/3, so that in each successive cycle, 1/3 of the remaining troops move out of the cell until it is empty. That fraction is adjusted with the**–mo ve** argument, 1 for less movement, and 10 for more movement. The option**–digin** <*n*> simulates the time and ef fort required for troops to dig in and build fortifications. This is achieved by reducing the rate of flow of troops into a cell as it fills up to capacity, so that to achieve a really full troop cell the men must dig in and settle down to accommodate the last arrivals. The argument <*n*> modulates the strength of this effect, from 1 to 10 for small to large. The maximum number of troops which can occupy a single cell is set via **–maxval** <*n*>. Note that for octagonal tiling only, the some cells (the square ones) will have different maxvals.

The **–nospigot** *[n]* option causes troops to automatically cease attacks when they are highly outnumbered, preventing the "spigoting" (perhaps "siphoning" would be more appropriate) which can empty whole supply lines into needless slaughter. Neighboring supply lines are shut off whenever the troops in a cell are outnumbered by a ratio given by the argument to the nospigot command.

BOARD CONFIGURATION OPTIONS

–cell, **–board**, **–boardx**, **–boardy**, **–border**, **–manpos**, **–xpos**, **–ypos**, **–area**, **–wrap**, **–grid**

The dimensions of the game board can be tailored via the **–boardx** <*n*> *and* **–boardy** <*n*> options which set the horizontal and vertical board dimensions, in terms of cells. The **–board** <*n*> option creates a square board. The dimension of each cell, in pixels, is set by the **–cell** <*n*> option. The xbattle window border can be set with **–border** <*n*> , while the initial x and y position of the game board can be set with **–xpos**

<*n*> and **−ypos** <*n*> respectively. The**−manpos** option allo ws each player to position his or her window interactively (does not work with all window managers). A grid indicating the borders of each cell is established via the **−grid** command (the default), and can be eliminated via the negative command **−no_grid**. Game play wraps around the edged of the board if the **−wrap** option is invoked, although certain tiling schemes require even or odd board dimensions for wrap to work properly in both the horizontal and vertical directions. Troop markers are scaled by area (proportional to number), rather than diameter, if the **−area** option is used.

BOARD TILING OPTIONS

−diamond, **−square**, **−hex**, **−octagon**, **−triangle**

A number of different tiling methods are available in **xbattle** , each of which employs cells of a different shape. Square cells in a rectangular grid are used for the**−squar e** option (the default). Hexagonal cells are used with the **−hex** option. The **−diamond** option results in a square tiling, tilted by 45 degrees. A tiling consisting of two orientations of equilateral triangles is invoked with the **−triangle** option. The **−octagon** option results in a tiling consisting of a combination of regular octagons and small squares. Since different cell shapes have different neighborhoods, troop movement in the different tilings can have a very different feel, and may take some getting used to.

DRAWING OPTIONS

−draw, **−new_colormap**

The method of drawing and erasing troops and terrain is defined via the **−draw** <*n*> option, where the argument indicates one of five distinct techniques, of varying speed and flicker. They are: Method 0: Erase the cell by drawing a circle the color of the terrain, then redraw the cell contents. This is the method employed in xbattle versions before 5.3. Although simple and fast, the onscreen erasing and redrawing often results in annoying flicker.

METHOD 1

Erase and redraw cells as for method 0, but do the whole process on a backing store pixmap, then copy the cell to the window. The copy from the pixmap to the window adds some overhead, but flicker is completely eliminated.

METHOD 2

Copy a blank cell from pixmap storage to a working pixmap, draw the cell contents, then copy the cell to the window. This method exchanges the cell erase of method 1 for a pixmap-to-pixmap copy operation to also provide flicker-free game animation. Unfortunately this method only works for square tilings, since only rectangular cells can be perfectly extracted during X-Window copy operations.

METHOD 3

Copy the cell from the window to a pixmap (along with a little surround for non-square cells), erase with a circle, redraw the contents, and then copy the cell back to the window. Like method 0, but with two extra copy operations and no flicker.

METHOD 4

Copy the cell from the window to a pixmap, erase the contents (including terrain) via an AND operation, draw the terrain via an OR operation, draw the cell contents, and copy back to the window. This method is fabulously complex, but has the advantage of being the only of the above method which redraws the entire cell contents, including terrain.

Method 0 is the default. With any reasonably fast-drawing machine, methods 1, 2, and 3 should provide quick-enough animation. Method 4 is a bit cumbersome. Which method is the fastest depends on how fast the display machine is at drawing lines and circles and at copying rectangles. Due to the buffering of X Window drawing commands, even method 0 provides reasonably good results since the cell erases often never appear on the screen before the cell redraw.

The use of a private colormap may be forced by specifying the **–new_colormap** option. The option will be ignored on non-paletted displays.

GUN AND PARATROOPER OPTIONS

–guns, **–guns_damage**, **–guns_cost**,
–para, **–para_damage**, **–para_cost**,
–manage

The command option **–guns** <*n*> enables the key 'g' to be used to control artillery, which can be shot from any occupied game cell. The range and direction of the shot are determined by the position of the cursor in the game cell relative to the center of the cell --- near center for short range and near the edge for long range, as modulated by the argument <*n*>. Every shell costs a number of troops from the source cell equal to the argument of **–guns_cost** <*n*> (default: 2), and destroys a number of troops at the destination cell equal to the argument of **–guns_damage** <*n*> (default: 1). The fall of shot is indicated by the brief appearance of a little dot of the attacker's color. With the **–horizon** option the fall of shot may not be visible for long range shots, although invisible enemy troops will be destroyed where the shell falls. Artillery can damage both friend and foe, so it must be used with caution. Paratroops are enabled by the option **–para** <*n*>, and are launched similarly to artillery using the 'p' key. The cost of dropping a number of troops equal to the argument of **–para_damage** <*n*> (default: 1) at the destination cell is equal to the argument of **–para_cost** <*n*> (default: 3). The drop zone is indicated by the brief appearance of a parachute symbol. When used with the **–manage** option, artillery and paratroops can be deployed continuously with the 'G' and 'P' keys instead of the 'g' and 'p' keys. This will initiate a continuous barrage that will only stop when the source cell is exhausted, but will recommence when it is resupplied. The managed command is indicated by the letters "GUN" or "PAR" in the source cell, and can be canceled with either the 'c' key, or by giving the source cell a movement command.

TERRAIN OPTIONS

–hills, **–hill_tones**, **–hill_color**,
–peaks, **–peak_bias**, **–trough_bias**,
–forest, **–forest_tones**, **–forest_color**,
–sea, **–sea_block**, **–sea_tones**, **–sea_color**, **–sea_value**

The command option **–hills** <*n*> initializes random hills which restrict movement when going from low to high elevation, and enhance movement from high to low, but do not affect movement on the level. The elevation is indicated by the shade of the cell, light for high and dark for low on monochrome, and brownish for high and greenish for low on color displays. The argument controls the amount of energy gained and lost on hills, i.e. the steepness. Hills provide a tactical advantage when attacking downhill. With very steep hills (**-hills** 9) movement from very low to very high elevation (a cliff) is virtually impossible. The number of discrete elevation levels is set via the **–hill_tones** <*n*> option. On color monitors, the hill hues can be tailored via the **–hill_color** <*n*> <*red*> <*green*> <*blue*>, where<*n*> specifies the ele vation index (from 0 to hill_tones-1) to be changed to the RGB triplet. The color of unspecified elevation indices are linearly interpolated based on specified indices.

The command option **–forest** <*n*> initializes random forests which restrict movement within the forest, but do not affect movement from thin to thick forest. On both color and monochrome displays, thick forest is dark, and thin is light. Forest may not be used in conjunction with hills. When transitioning from one forest thickness to another, the movement is determined by the destination cell, not the source cell, so that troops deployed within a forest but at the boundary have a tactical advantage over those deployed outside the boundary. As for hills, the number of distinct forest densities is specified via the**–f orest_tones** <*n*>option, with colors being specified by the **–forest_color** <*n*> <*red*> <*green*> <*blue*> option.

The command option **–sea** <*n*> generates randomly distributed bodies of water, whose prevalence is determined by the argument <*n*>. Such bodies of water cannot be crossed by infantry. A small value creates scattered ponds and lakes, which influences the tactical deployment of troops, whereas a large value creates a maze-like pattern of fjords or rivers which isolate blocks of land into islands which can only be taken by paratroops. On monochrome monitors water appears dark mottled grey, and on color monitors it appears as various shades of blue. Like hills, seas have elevation (depths), the number of which is controlled via the **−sea_tones** <*n*> option, with colors determined by the **−sea_color** <*n*> <*red*> <*green*> <*blue*> option. Besides looking nice, sea depths are useful when playing with the **−dig** and **−fill** options (see the TERRAIN MODIFICATION OPTIONS section). On monochrome monitors, the option **−sea_value** <*float*> determines the blackness of the shallowest sea, expressed as a fraction. For backwards compatibility, sea depths can also be indicated by the size of the sea marker if the **−sea_block** option is invoked.

Hills (and forests and seas) are created by a complex terrain generation algorithm which bases elevations (or densities, in the case of forests) on a number of fixed points, as specified by the **−peaks** <*n*> option. A non-linear interpolation process determines the elevations of the rest of the game cells based on these randomly-determined positions and elevations. The**−peak_bias** <*float*> option determines ho w hill elevations and forest densities will be distributed --- 0.0 yields generally low-elevation terrain, with spire-like mountains, while 2.0 yields generally high-elevation terrain, with deep ravines. The default value of 1.0 results in pleasantly contoured terrain. Similarly, the **−trough_bias** <*float*> option controls the distribution of sea depths.

TERRAIN MODIFICATION OPTIONS

−dig, **−dig_cost**,
−fill, **−fill_cost**,
−build, **−build_cost**, **−build_limit**,
−scuttle, **−scuttle_cost**,
−manage

The command options **−dig** *[n]* and **−fill** *[n]* allow run time modification of the terrain by digging hills and seas down to lower elevation or filling them up to higher elevation. This allows the construction and breaching of defensive fortifications. The cost of these operations (in troops) is determined by the **−dig_cost** <*n*> and **−fill_cost** <*n*> options. The operations are accomplished by positioning the mouse on the friendly cell and striking the "d" key (for dig) or the "f" key (for fill). With the **−sea** option, **−dig** <*n*> and **−fill** <*n*> can be supplied with an argument which specifies the number of sea depths (see also **−sea_tones**). Since it is impossible to occupy a sea cell to fill it, filling seas is accomplished by setting the command vector as if to move into the sea, and then pressing "f". Likewise for digging a sea deeper. For all other fill and dig operations the troop cell may not have any command vectors set.

The **−build** <*n*> and **−scuttle** *[n]* options allow the building and destruction of bases (or towns). The costs of these operations (in troops) are determined by **−build_cost** <*n*> and **−scuttle_cost** <*n*>. When the mouse is positioned on a friendly cell and the "b" key is pressed, the troops are exchanged for a 1/<*n*> fraction of a base, displayed as an arc segment. Thus<*n*> b uilding commands are required to produce a functional base. When the capture of a base by the enemy seems inevitable, it is often advisable to scuttle the base to prevent it falling into enemy hands. Scuttling is performed by positioning the mouse on the base and pressing the "s" key. If the build option is not enabled, this reduces the size (and production capacity) of that base by one scuttle unit for each scuttle_cost of troops expended, where a scuttle unit is defined by the argument of the scuttle option (default: 5). Usually, several keystrokes are required to complete the destruction. When used in conjunction with the**−b uild** option, instead of reducing the size of the base, each scuttle operation removes a section (arc segment) of the base, at a troop cost indicated by the **−scuttle_cost** <*n*> option. A base will not produce troops if even a single segment is missing, although of course it is less expensive to repair (with "b" build) a base with fewer segments missing.

As with **−guns** and **−para**, the **−dig**, **−fill**, and **−build** options (but not the **-scuttle** option) can be "managed" with the **−manage** option, allowing a player to issue a single command to initiate a sequence of repeated dig, fill, or build operations using the keys ’D’, ’F’, and ’B’ respectively. The managed operation will continue until the task is done, as long as the cell is resupplied. The managed operation will be indicated by the letters "DIG", "FIL" or "BLD" respectively in the managed cell. Managed operation can be canceled with the ’c’ key, or by issuing a movement command to the cell.

VISIBILITY OPTIONS

−horizon, **−hidden**, **−map**, **−basemap**, **−localmap**

The command option **−horizon** *[n]* restricts the view of enemy troop deployment to within<*n*> cells of any friendly troops. Horizon can be called with no argument, in which case the default value of 2 is used. Intelligence of more remote regions can be gathered by use of paratroops. The command option **−hidden** (no arguments) makes the command vectors of the enemy invisible at any range. The command option **−map** is similar to **−horizon** except that it restricts your view of geographical objects as well as enemy troops, although it will "remember" any terrain that you have seen once, as if you had mapped that information. The**−basemap** option maps bases and to wns as it does the terrain --- once you see them, they’re remembered. The option**−localmap** maps only the local area around your troops, and features disappear as you move away again.

VICTORY DETECTION OPTIONS

−win_trad, **−win_army**, **−win_land**, **−win_timeout**, **−win_pos**,
−win_wait

The victory options instruct the game to look for one or more victory conditions which will cause the game to end. The exit status of the program will indicate which player won (see EXIT STATUS below).

The **−win_trad** option specifies that the game is won if the other sides have no toops on the map. This may require searching the whole map for just a few remaining toops so the **−win_army** <*float*> option may be used to only require this side to control<*float*> percent of all armies on the map. The **−win_land** <*float*> option will allow victory if<*float*> percent or more of the map is occupied by this side’ s troops (sea cells are also counted).

Victory may be granted after a period of time with the **−win_timeout** <*int*> option. The timeout is specified in seconds. This is useful for campaigns where one side plays a defensive role.

Victory can be granted if a specific location on the board is occupied with the **−win_pos** <*x*> <*y*> option. The arguments are specified in cells. More than one position may be specified by using the option more than once. The **−win_wait** <*int*> option specifies a delay in seconds between victory detection due to any other option and the end of the game. This is set to 10 seconds by default.

FIXME: player or side? Victory conditions can be set per player or for all players (see BIASED GAMES below). If several conditions are set for the same player, the player will win if any of them is satisfied. For instance:

```
Any player wins if no other players remain:
xbattle ... -win_trad

Exit game after 200 turns (there will be a draw if multiple players remain):
xbattle ... -win_timeout 200

Player 1 wins by reaching the upper left corner, player 2 wins by
keeping him out for 300 turns:
xbattle ... -red { -win_pos 0 0 } foo:0 -blue { -win_timeout 300 } bar:0

Like the previous example, but both sides can also win by exterminating
the opponent:
xbattle ... -red { -win_pos 0 0 } foo:0 -blue { -win_timeout 300 } bar:0 -win_trad
```

CLIENT/SERVER OPTIONS

−is_server, **−use_server**

A client/server architecture has been added to allow multiplayer games over networks that are too slow to allow realtime X-forwarding. A game of **xbattle** can now consist of one or several running instances of the program on different computers. One of the computers must be designated the server and it can have one or more players connected to it in the normal way using multiple X11 displays. The other players will be connected as clients. Since the clients maintain their own displays the need for a fast connection between the players is not as big.

When starting a game on several computers, the server should be started first. The server should use the **−is_server** <*port*> option for any players which will be connected as clients. Use a display name of **null** or **you** for these players. All other options should be specified as normal. If multiple server options are used, they must use different port numbers. On most systems, regular users are only allowed to use ports higher than 1024. The **−is_server** option should never be used as an all-player argument because it will assign the same port number to all players.

Each remote player should start **xbattle** with exactly the same options as specified on the server with three exceptions. The players options should not include the**−is_ser ver** directive and any non-local players should have a display or **null** or **you**. Additionally, **−use_server** <*host*> <*port*> <*ply*> option is needed to specify which server to use. The port number should be the same one as specified for this player with the **−is_server** option. Clients should not repeat any **−ai** options used on the server because this will lead to having two copies of the AI controlling the same player. Some examples are included below to clarify the use of these options.

To start a game with two human players, one directly on the server and one connection with a client, you would run this command on the server ...

```
xbattle -red :0 -blue { -is_server 8000 } null -rbases 3 -hills 5 -sea 3
```

and this command on the client ...

```
xbattle -red null -blue :0 -use_server Ada 8000 -rbases 3 -hills 5 -sea 3
```

To start a game with four players, two humans directly connected to the server, one human connecting with a client, and one AI on the server you would run this command on the server ...

```
xbattle -red :0 -blue null -green { -is_server 8000 } null
    -yellow Ceasar:0.0 -ai 1 1 0.5 -rbases 3 -hills 5 -sea 3
```

and this command on the client ...

```
xbattle -red null -blue null -green :0.0 -use_server servername 8000
    -yellow null -rbases 3 -hills 5 -sea 3
```

The client could have connected as a second AI even though the server did not specify it ...

```
xbattle -red null -blue null -green null -use_server servername 8000
    -yellow null -rbases 3 -hills 5 -sea 3 -ai 2 1 0.5
```

The difference between player 1 (blue) and player 2 (green) is that the AI for player 1 runs on the server whereas the AI for player 2 runs on the client.

STORE AND REPLAY OPTIONS

−store, **−replay**

The **−store** <*file*> option allows you to store enough information about the visual progress of the game to reconstruct it later with **−replay** <*file*> option. When **−replay** is used, all other command options are ignored except the **-**<*color*> <*display*> options, which can be used to send the replay to other displays. When doing so, only the<*display*> portion of the option is used, the <*color*> is ignored. So, if you play a game with many command line parameters and several displays with the argument **−store** <*file*>, after the game you can repeat the same command line but just change **−store** to **−replay**, and the game will be replayed on the displays of all the original combatants. When xbattle is called with the **−replay** option alone, the default display will be "me". If store or replay are called without a file name, the default name *xbattle.xba* will be used. In the replay, the view restrictions of the **−horizon** option are deactivated, i.e. all enemy troops are visible. The replay action can be paused or resumed by typing any key, and can be interrupted with either control-c or control-q.

GAME STATE SAVING, LOADING, AND EDITING OPTIONS

−load, **−dump**, **−overwrite**, **−edit**

The game state can be saved at any point during the game with the control-p key. This creates a file called *xbattle.xbt*, or the name given with the argument **−dump** <*filename*>, which represents the state of the game board at the time of saving. Future games can be started from the saved game state with the command option **−load** <*file*> where <*file*> is optional if the file name is **xbattle.xbt**. If the specified load file cannot be found in the current directory, xbattle will search the default xbt directory DEFAULT_XBT_DIR, which can be specified at compile time in the makefile. Note that most game parameters ARE NOT STORED. Only terrain features (forest, hills, seas, towns etc.) and troop deployment. This means that if you were playing with **−farms**, **−decay**, and **−guns** then you will have to type them in if you want them for the new game. The terrain and boardsize of the saved map file will override all terrain and boardsize arguments when loaded. Troop and town/base producing options (such as **−militia**, **−towns**, and **−rbases**) will add new features on top of the loaded game state. If the **−overwrite** option is issued, only the terrain and cities from the loaded game will be used --- no troops will appear. This is useful for repeating games with interesting terrains with different troop configurations.

Game boards can be created or modified with the edit function, which is called with the command option **−edit** <*file*> where <*file*> is optional if the file name is **xbattle.xbt**. With this option, no game is played, but instead, the mouse and key commands control the features of the map to be edited. To edit an existing file, use **−edit** <*file*> and type "l" when the editor comes up. This will load the file named in the edit argument. To save that file, type "d" and the file will be saved to the same file name. No provision is made for saving to a different file name. When using the edit mode, the command line arguments must reflect the number and color of players to be used, and the sea, forest or hills options if they will be required. For example, to create a map called "mymap.xbt" with three color teams and seas, could use the command ...

```
xbattle -edit mymap.xbt -sea 7 -white me -black you -dark you
```

Note the use of the special display "you", which is a dummy display name used as a place holder for the black and dark colors. The interactive commands are as follows:

```
left button:    lower terrain by one notch (sea lowest)
middle button:  raise terrain by one notch
right button:   toggle between lowest and mid terrain

c:    create city (growth = 100)
t:    create town (growth = 80)
v:    create village (growth = 60)
k:    increase size of city by 5 percent
```

j: decrease size of city by 5 percent
s: scuttle city - remove 36 degrees of arc
b: build city - add 36 degrees of arc

o: create a forest cell
i: change the base side for this cell

0-9: create troop marker with troops of current color
[: decrease troops by 1
]: increase troops by 1
r: increment current color
f: change color of existent troop marker
d: dump board with name <filename>
l: load board with name <filename>
q: quit

With the **–edit** option, the **–overwrite** option has a slightly different function. Rather than suppress the display of troops, as it does when combined with the **–load** option, the **–overwrite** option causes default terrain to be generated for editing. Note that boards created with during the edit process are stored in a reduced format, whereas boards saved during game play are stored in standard format, which includes more information about each cell, at the cost of about 15 times more storage space. Standard format files can also be edited.

INTERACTIVE CONTROLS (MOUSE AND KEYBOARD)

Movement commands are performed with the left and middle mouse buttons, to direct the command vector. A click in the center of the game cell clears all command vectors; a click near an edge sets the vector in that direction, and a click near a corner sets the two adjacent vectors. The left mouse toggles command vectors while the middle mouse clears existing vectors and sets a new vector (An alternative command system is available, see COMPILATION NOTES below). The right mouse is used to repeat the last used command (with the **–repeat** option). The keyboard is interpreted differently depending on whether the mouse is positioned on the gameboard or on the text area below. On the gameboard, the the keys control-s and control-q pause and resume the game respectively. The 'z' key cancels all command vectors to the cell containing the cursor (like a click in the center of the cell). The key control-p saves the current game to a map file (see Saving Game State commands below). There are also a variety of keyboard commands available with different options, to control special functions on the gameboard. These keystrokes are described in detail with the description of the appropriate options (see **–guns**, **–para**, **–build**, **–scuttle**, **–fill**, **–dig**, **–reserve**). In the text area below the keyboard, the keys control-c and control-q both exit the player from the game, although the game continues among the remaining players until they also quit, and the key control-w also exits the player, but allows him or her to continue watching as the other players play on. The rest of the keyboard is used for communication between participants through text lines. This is especially useful when playing between remote sites -- each team has its own text line, and the color of the text matches the color of the team. The control-g key rings the bell on all displays, which can be used to draw attention to a new message. All keyboard commands can be reconfigured by changing the pairings in the keyboard.h file and recompiling.

BIASED GAMES

The game can be biased to favor a less experienced player, or for any other reason, in the following way. In the normal syntax, the command line argument "-<color>" is immediately followed by the "<display>" argument, for example "-black me". It is possible to define command line options that are specific to only one player with the syntax "-<color> { <options> } <display>" where <options> refers to a list of command line options as before, but is included in a set of braces between the team color and the display (note the spaces on either side of the braces). For example,

```
xbattle -black { -fight 10 } me -white { -fight 5 } cnsxk
```

where black (on display "me") has the advantage of greater firepower than white (on display "cnsxk"). Not all options can be biased, specifically options that control the global behavior of the game, such as **−speed**, **−hex**, and **−board**. Note also that if you are using player-specific and global options, the global options MUST be listed first, otherwise they will overwrite the player-specific options. For example,

```
  xbattle -black { -fight 10 } me -white cnsxk -fight 5
```

will result in a fight 5 for both players. In order to achieve the desired result, the command line must be...

```
 xbattle  -fight 5 -black { -fight 10 } me -white cnsxk
```

where the local option overwrites only the black team's fight value.

EXTENSIONS

A great deal of effort has been made in the design of this game to make it as simple and modular as possible. Please send any interesting variations or extensions to lesher@cns.bu.edu.

EXAMPLES

Here are some example games to give an idea of the variability of the parameters. The first example is a simple symmetrical game between "me" in black on my own display, and a white opponent on the display "cnsxk:0.0". The troops will be rapidly exhausted in this small skirmish.

```
  xbattle -black me -white cnsxk:0.0 -armies 4
```

The next example adds bases, which will produce a much prolonged conflict involving long supply lines between the front and the bases, much like desert warfare. One conflict in this battle represents a skirmish like the entire game of the previous example. In this example black is playing on the display cnsxk:0.0, and white is on the system console. Note that the extension ":0.0" can be omitted.

```
  xbattle -black cnsxk -white unix -armies 4 -bases 2
```

The next example is a game with militia scattered around initially, that have to race to occupy the towns and link up with their compatriots before they can eliminate the enemy. This is a dynamic scenario requiring tactical and strategic skill and fast reflexes. In this example black is playing on cnsxk:0.0 while white is playing on the system console of the remote machine thalamus.bu.edu.

```
  xbattle -black cnsxk -white thalamus.bu.edu -towns 2
         -militia 2 -hills 7
```

Here is a favorite around B.U. where the land is broken up by many bodies of water creating isolated islands, and view of the enemy is restricted to nearby cells, resulting in lots of surprises. Paratroops can be used for reconnaissance by launching them into unknown sectors, and they must be used in conjunction with heavy artillery barrages for airborne assaults from one landmass to the next. In this example the color display will show cyan and red teams, while the monochrome monitor will show white and black teams respectively. The decay option prevents huge armies from building up at the end of the game, and the -store option is used to store this game to the file "xbattle.xba".

```
  xbattle -cyan_white thalamus:0.0 -red_black cnsxk
         -rbases 5 -sea 8 -guns 4 -para 4 -horizon 2
         -decay 3 -store xbattle.xba
```

Now, the previous stored game is replayed to the original displays by repeating the original command line except that -store is changed to -replay. This is convenient if you have command line editing facilities.

```
xbattle -cyan_white thalamus:0.0 -red_black cnsxk
        -rbases 5 -sea 8 -guns 4 -para 4 -horizon
        -replay xbattle.xba
```

With -replay, all arguments are actually ignored except the displays, so you could achieve exactly the same result with the simpler command

```
xbattle -black thalamus:0.0 -black cnsxk -replay
```

where the -black argument flags the subsequent argument as a displayname, but is otherwise ignored, i.e. any color name would suffice. The filename for -replay is omitted, so that the default file name "xbattle.xba" is used.

The next example illustrates the use of the options file, xbos/tribal.xbo, to set up a game including, decay, seas, farms, militia, and many other options.

```
xbattle -black me -white thalamus -options xbos/tribal.xbo
```

Options files can also be read in individually for the two players, as in the following example...

```
xbattle -options game.xbo -black me
        -white { -options xbos/weak.xbo } thalamus
```

This results in a biased game where both black and white receive the options defined in game.xbo, and white receives some specific handicaps defined in weak.xbo. For example, weak.xbo could define 2 rbases instead of 5, horizon of 1 instead of 2, and lower movement and fighting values. Since these options overwrite existing options in game.xbo, the command line arguments may NOT be typed in arbitrary order. Global options must be defined before they are overwritten by the specific options to the white team.

SAMPLE .XBO AND .XBT FILES

To provide some idea of the range of gameplay available with xbattle, a number of option files (.xbo extension) and dump files (.xbt extension) are provided with the xbattle 5.4.1 release in the "xbos" and "xbts" subdirectories, respectively. These are listed below, along with very brief descriptions.

tribal.xbo	mad scramble, every man for himself
skirmish.xbo	intrigue, espionage, plotting
demo.xbo	demo which includes ALL options
atlas.xbo	standard atlas terrain/color scheme
desert.xbo	mountainous desert terrain/color scheme
io.xbo	Io-like terrain/color scheme
space.xbo	space-like terrain/color scheme
tropical.xbo	tropical islands terrain/color scheme
tundra.xbo	tundra-like terrain/color scheme
castle.xbt	moated fortress with villages
natural.xbt	natural streams, lakes, and hills

PLAYING TIPS

The first thing you must learn is to click quickly and accurately on the game cells. Do not focus your attention on one region of the board, but scan the whole board frequently. Look at the big picture- capture the towns that will support each other, especially a well positioned cluster of big towns. Eliminate all enemy troops from your rear, and advance outwards, preferably from a corner, with a well supplied front. Travel in convoy for speed and efficiency in safe regions, especially if you are playing with

-decay, but fan out near the enemy to provide alternate routes to a broad front (click on the corner to open two command vectors simultaneously). Avoid head-on assaults on the enemy, but rather dig in and wait for him to attack while you try to turn his flank and cut off his supplies to the front, or concentrate at his weakest points. When advancing, try to attack weak cells with strong ones to gain maximum advantage, and be alert for losing battles of your weak cells pouring into a strong enemy cell, which will drain your resources until you cancel the attack and build up reserves. If however you are fighting a delaying action, or retreating under fire then you should attack strong enemy cells with your weak ones on a broad front to conserve resources. This is particularly effective with the -disrupt option. Always try to attack a cell from two or more sides, and build up sufficient strength before launching an attack on a strong cell. Always consider the "manufacturing capacity" of the enemy, i.e. the number and size of bases and towns, as the one with the most capacity will eventually win. Watch out for single enemy commandos near your unprotected bases, especially when playing with paratroops, and use such commandos to good effect against an inattentive opponent. You can keep a base fortified while sending troops to the front by use of recurrent connections, going in loops or in both directions, or by establishing dead-end branches along the supply line to accumulate local reserves. You should always have a few strong reserves near your base when playing with -horizon or -para, to ensure against surprise attacks. When playing with horizon and paratroops use the paratroops to gather intelligence from beyond the horizon. When playing with paratroops or artillery, you can create a network of recurrent connections near the bases that will provide protection by automatically sending troops into parts of the net that are knocked out.

EXIT STATUS

If an error (memory alloction, bad command-line, missing files, etc.) occurs the exit status will be 1. If victory detection is not enabled, successful completion is indicated with an exit status of 0. If victory detection is enabled, the exit status will be 10 plus the side id of the victor. The first side will return 10, the second 11, and so forth. If multiple sides are victorious, the exit status will be 9. This can happen if the same timeout is used for all sides.

COMPILATION NOTES

Certain other game options or alternatives are allowed at compile time by editing the file "constant.h" before compiling the program. This file contains many global flags which can be set to TRUE or FALSE. The FIXED_INVERT flag may be set to FALSE if you do not like the appearance of the inverted command vector within the troop cell. The FIXED_VARMOUSE option may be set to TRUE if you would like the mouse operations to be redefined so that the left mouse adds command vectors, and the middle mouse subtracts such vectors. The flag FIXED_PAUSE may be set to FALSE to disable the ability to pause and resume the game with control-s and control-q. The FIXED_SHOWFLOW flag in extern.h may be set to FALSE to make the displayed command vectors remain at full length even when the troop strength is zero. The flag FIXED_MULTITEXT can be set to FALSE, whereby instead of having a single text line for each player, two text lines are shared by all the players. The flag FIXED_MULTIFLUSH can be set to TRUE, whereby command vectors appear immediately after the command is given, although performance is noticeably impaired. If a player repeatedly "nukes" the whole game when he is losing, you can set FIXED_FATAL_RECOVER to TRUE in to enable recovery. You may choose between FIXED_USE_LONGJMP and FIXED_USE_PROCEDURE recovery methods if FIXED_FATAL_RECOVER is set true. The former uses the Unix functions setjmp() and longjmp(). The latter uses a normal procedure call to recover. We recommended use of LONGJMP. After 20 fatal errors program kicks out, as a failsafe. WARNING: use FATAL_RECOVER at your own risk. If the flag FIXED_TIMER is set set to TRUE, the elapsed time from game startup will be displayed in the lower left-hand corner of the screen, within the text pane.

BUGS

When the system is slow, there is a noticeable time lag between the mouse positioning and the keystroke registration, so that a keystroke for a cell pointed to by the mouse might be actually recorded in the next cell the mouse moves to. Similarly, a shifted mouse click (as for paratroops) might be delayed so that by the time it is processed the shift key is no longer being depressed, and it is recorded as an unshifted mouse click (as for artillery). Under such circumstances, avoid issuing rapid command sequences. Remote play is extremely difficult. When a {player-specific option} is followed by a universal option, the former is overwritten by the latter, so the {player-specific option} should always follow the universal option.

AUTHORS

Greg Lesher (lesher@cns.bu.edu), Steve Lehar (slehar@park.bu.edu), and some sections of code from Mark Lauer (elric@basser.cs.su.oz.au). Helpful suggestions, bug reports, and ideas were gratefully received from numerous contributors from all over the world.