sun®
microsystems

SunOS Reference Manual

Part Number: 800-3827-10
Revision A of 27 March, 1990

NAME

intro – introduction to games and demos

DESCRIPTION

This section describes available games and demos.

LIST OF GAMES AND DEMOS

Name	Appears on Page	Description
adventure	**adventure**(6)	an exploration game
arithmetic	**arithmetic**(6)	provide drill in number facts
backgammon	**backgammon**(6)	the game of backgammon
banner	**banner**(6)	print large banner on printer
battlestar	**battlestar**(6)	a tropical adventure game
bcd	**bcd**(6)	convert to antique media
bdemos	**bdemos**(6)	demonstrate Sun Monochrome Bitmap Display
bdraw	**draw**(6)	interactive graphics drawing
bj	**bj**(6)	the game of black jack
boggle	**boggle**(6)	play the game of boggle
boggletool	**boggletool**(6)	play a game of boggle
bouncedemo	**graphics_demos**(6)	graphics demonstration programs
brotcube	**brotcube**(6)	rotate a simple cube
bsuncube	**bsuncube**(6)	view 3-D Sun logo
buttontest	**buttontest**(6)	demonstration and testing program for SunButtons
canfield	**canfield**(6)	Canfield solitaire card game
canfieldtool	**canfield**(6)	Canfield solitaire card game
canvas_demo	**sunview_demos**(6)	Window-System demonstration programs
cdplayer	**cdplayer**(6)	CD-ROM audio demo program
cdraw	**draw**(6)	interactive graphics drawing
cfscores	**canfield**(6)	Canfield solitaire card game
chess	**chess**(6)	the game of chess
chesstool	**chesstool**(6)	window-based front-end to chess program
ching	**ching**(6)	the book of changes and other cookies
colordemos	**colordemos**(6)	demonstrate Sun Color Graphics Display
craps	**craps**(6)	the game of craps
cribbage	**cribbage**(6)	the card game cribbage
cursor_demo	**sunview_demos**(6)	Window-System demonstration programs
dialtest	**dialtest**(6)	demonstration and testing program for SunDials
draw	**draw**(6)	interactive graphics drawing
factor	**factor**(6)	factor a number, generate large primes
fish	**fish**(6)	play ''Go Fish''
flight	**gp_demos**(6)	demonstration programs for the Graphics Processor
fortune	**fortune**(6)	print a random, hopefully interesting, adage
framedemo	**graphics_demos**(6)	graphics demonstration programs
gaintool	**gaintool**(6)	audio control panel
gammontool	**gammontool**(6)	play a game of backgammon
gp_demos	**gp_demos**(6)	demonstration programs for the Graphics Processor
graphics_demos	**graphics_demos**(6)	graphics demonstration programs
hack	**hack**(6)	replacement for rogue
hangman	**hangman**(6)	computer version of the game hangman
hunt	**hunt**(6)	a multiplayer multiterminal game
jumpdemo	**graphics_demos**(6)	graphics demonstration programs
life	**life**(6)	John Conway's game of life
mille	**mille**(6)	play Mille Bornes
monop	**monop**(6)	Monopoly game

moo	**moo**(6)	guessing game
number	**number**(6)	convert Arabic numerals to English
play	**play**(6)	play audio files
ppt	**bcd**(6)	convert to antique media
primes	**factor**(6)	factor a number, generate large primes
primes	**primes**(6)	print all primes larger than some given number
quiz	**quiz**(6)	test your knowledge
rain	**rain**(6)	animated raindrops display
random	**random**(6)	select lines randomly from a file
raw2audio	**raw2audio**(6)	convert raw audio data to audio file format
record	**record**(6)	record an audio file
robots	**robots**(6)	fight off villainous robots
rotcvph	**rotcvph**(6)	rotate convex polyhedron
rotobj	**gp_demos**(6)	demonstration programs for the Graphics Processor
snake	**snake**(6)	display chase game
snscore	**snake**(6)	display chase game
soundtool	**soundtool**(6)	audio play/record tool
spheresdemo	**graphics_demos**(6)	graphics demonstration programs
suncoredemos	**suncoredemos**(6)	demonstrate SunCore Graphics Package
sunview_demos	**sunview_demos**(6)	Window-System demonstration programs
trek	**trek**(6)	trekkie game
vwcvph	**vwcvph**(6)	view convex polyhedron
worm	**worm**(6)	play the growing worm game
worms	**worms**(6)	animate worms on a display terminal
wump	**wump**(6)	the game of hunt the wumpus

NAME

adventure – an exploration game

SYNOPSIS

/usr/games/adventure

DESCRIPTION

The object of the game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type **quit**; to save a game for later resumption, type **suspend**.

BUGS

Saving a game creates a large executable file instead of just the information needed to resume the game.

NAME

arithmetic – provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [+–x/] [*range*]

DESCRIPTION

arithmetic types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (such as CTRL–C).

The first optional argument determines the kind of problem to be generated; '+', '–', 'x', '/' respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +–.

range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

backgammon – the game of backgammon

SYNOPSIS

backgammon [–] [**n r w b pr pw pb t***term* **s***filename*]

DESCRIPTION

backgammon lets you play backgammon against the computer or against a 'friend'. All commands only are one letter, so you don't need to type a carriage return, except at the end of a move. **backgammon** is mostly self documenting, so that a q **?** (question mark) will usually get some help. If you answer y when **backgammon** asks if you want the rules, you will get text explaining the rules of the game, some hints on strategy, instruction on how to use **backgammon,** and a tutorial consisting of a practice game against the computer. A description of how to use **backgammon** can be obtained by answering y when it asks if you want instructions. The possible arguments for **backgammon** (most are unnecessary but some are very convenient) consist of:

n	don't ask for rules or instructions
r	player is red (implies n)
w	player is white (implies n)
b	two players, red and white (implies n)
pr	print the board before red's turn
pw	print the board before white's turn
pb	print the board before both player's turn
t*term*	terminal is type *term*, uses */etc/termcap*, otherwise uses the TERM environment variable.
s*filename*	recover previously saved game from *filename*. This can also be done by executing the saved file, that is, typing its name in as a command.

Arguments may be optionally preceded by a – sign. Several arguments may be concatenated together, but not after **s** or **t** arguments, since they can be followed by an arbitrary string. Any unrecognized arguments are ignored. An argument of a lone – gets a description of possible arguments.

If **term** has capabilities for direct cursor movement. **backgammon** 'fixes' the board after each move, so the board does not need to be reprinted, unless the screen suffers some horrendous malady. Also, any 'p' option will be ignored.

QUICK REFERENCE

When **backgammon** prompts by typing only your color, type a space or carriage return to roll, or

d	to double
p	to print the board
q	to quit
s	to save the game for later

When **backgammon** prompts with 'Move:', type

p	to print the board
q	to quit
s	to save the game

or a *move,* which is a sequence of

s-f	move from s to f
s/r	move one man on s the roll r separated by commas or spaces and ending with a newline. Available abbreviations are

s-f1-f2	means **s-f1,f1-f2**
s/r1r2	means **s/r1,s/r2**

Use **b** for bar and **h** for home, or **0** or **25** as appropriate.

FILES

/usr/games/teachgammon	rules and tutorial
/etc/termcap	terminal capabilities

BUGS

backgammon's strategy needs much work.

NAME

banner – print large banner on printer

SYNOPSIS

/usr/games/banner [–w*n*] message . . .

DESCRIPTION

banner prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If –w is given, the output is reduced from a width of 132 to *n*, suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

BUGS

Several ASCII characters are not defined, notably '<', '>', '[', ']', '\', '^', '_', '{', '}', '|', and '~'. Also, the characters '"', ''', and '&' are funny looking (but in a useful way.)

The –w option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

NAME

battlestar – a tropical adventure game

SYNOPSIS

battlestar [**–r**]

DESCRIPTION

battlestar is an adventure game in the classic style. However, it is slightly less of a puzzle and more a game of exploration. There are a few magical words in the game, but on the whole, simple English should suffice to make one's desires understandable to the parser.

OPTIONS

–r Recover a saved game.

THE SETTING

In the days before the darkness came, when battlestars ruled the heavens...

Three He made and gave them to His daughters,
Beautiful nymphs, the goddesses of the waters.
One to bring good luck and simple feats of wonder,
Two to wash the lands and churn the waves asunder,
Three to rule the world and purge the skies with thunder.

In those times great wizards were known and their powers were beyond belief. They could take any object from thin air, and, uttering the word '**su**', could disappear.

In those times men were known for their lust of gold and desire to wear fine weapons. Swords and coats of mail were fashioned that could withstand a laser blast.

But when the darkness fell, the rightful reigns were toppled. Swords and helms and heads of state went rolling across the grass. The entire fleet of battlestars was reduced to a single ship.

USAGE

Sample Commands

```
take     ---     take an object
drop     ---     drop an object
wear     ---     wear an object you are holding
draw     ---     carry an object you are wearing
puton    ---     take an object and wear it
take off ---     draw an object and drop it
throw  <object> <direction>
!        <shell esc>
```

Implied Objects

```
>-: take watermelon
watermelon:
Taken.
>-: eat
watermelon:
Eaten.
>-: take knife and sword and apple, drop all
knife:
Taken.
broadsword:
Taken.
apple:
Taken.
knife:
Dropped.
```

```
broadsword:
Dropped.
apple:
Dropped.
>-: get
knife:
Taken.
```

Notice that the "shadow" of the next word stays around if you want to take advantage of it. That is, saying '**take knife**' and then '**drop**' will drop the knife you just took.

Score and Inven

The two commands **score** and **inven** will print out your current status in the game.

Saving a Game

The command **save** will save your game in a file called **Bstar**. You can recover a saved game by using the **–r** option when you start up the game.

Directions

The compass directions N, S, E, and W can be used if you have a compass. If you do not have a compass, you will have to say **R**, **L**, **A**, or **B**, which stand for Right, Left, Ahead, and Back. Directions printed in room descriptions are always printed in R, L, A, & B relative directions.

BUGS

Countless.

NAME

bcd, ppt – convert to antique media

SYNOPSIS

/usr/games/bcd *text*

/usr/games/ppt

DESCRIPTION

bcd converts the literal *text* into a form familiar to old-timers.

ppt converts the standard input into yet another form.

SEE ALSO

dd(1)

NAME

bdemos – demonstrate Sun Monochrome Bitmap Display

SYNOPSIS

/usr/demo/bballs
/usr/demo/bbounce
/usr/demo/bdemos
/usr/demo/bjump
/usr/demo/bphoto *file*
/usr/demo/brotcube

DESCRIPTION

Bdemos is a collection of simple demonstration programs for the Sun Monochrome Bitmap Display. Each program is briefly described below. Unless otherwise noted, each program should be terminated by typing the appropriate key (usually DELETE or ^C) to generate an interrupt signal.

bballs	colliding balls demo
bbounce	bouncing square demo
bdemos	a collection of demos This program has a menu for selection of several different demos. After typing a key to select a particular demo, the user may type ^C to get back the menu. Type 'q' to quit.
bjump	simulated jump to hyperspace
bphoto *file*	dither monochrome image *file* to bitmap display Image files suitable for display by this program are in */usr/demo/bwpix*.
brotcube	black and white spinning cube

FILES

/usr/demo/bwpix

SEE ALSO

bsuncube(6), **draw**(6)

NAME

bj – the game of black jack

SYNOPSIS

/usr/games/bj

DESCRIPTION

bj is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

> The bet is $2 every hand.
>
> A player ''natural'' (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a ''push'' (no money exchange).
>
> If the dealer has an ace up, the player is allowed to make an ''insurance'' bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.
>
> If the player is dealt two cards of the same value, he is allowed to ''double''. He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)
>
> If a dealt hand has a total of ten or eleven, the player may ''double down''. He may double the bet ($2 to $4) and receive exactly one more card on that hand.
>
> Under normal play, the player may ''hit'' (draw a card) as long as his total is not over twenty-one. If the player ''busts'' (goes over twenty-one), the dealer wins the bet.
>
> When the player ''stands'' (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.
>
> If both player and dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following questions will be asked at appropriate times. Each question is answered by y followed by a new-line for ''yes'', or just new-line for ''no''.

> **?** (this means, ''do you want a hit?'')
> **Insurance?**
> **Double down?**

Every time the deck is shuffled, the dealer so states and the ''action'' (total bet) and ''standing'' (total won or lost) is printed. To exit, hit the interrupt key (CTRL–C) and the action and standing will be printed.

NAME

boggle – play the game of boggle

SYNOPSIS

/usr/games/boggle [+] [++]

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

NAME

boggletool – play a game of boggle

SYNOPSIS

/usr/games/boggletool [*number*] [+[+]] [16-character *string*]

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

boggletool allows you to play the game of Boggle (TM Parker Bros.) against the computer. The *number* argument specifies the time limit in minutes (the default is 3 minutes). If a 16 character long string is placed on the command line, it is interpreted as a Boggle board: the first four letters form the top row, the next four letters the second row, etc. If no letters are specified, a board is randomly rolled by the computer from a set of Boggle cubes. The +[+] argument is explained below under **Advanced Play** .

PLAYING THE GAME

Rules of the Game

The object of Boggle is to find as many words as possible in a 4 by 4 grid of letters within a certain time limit. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. Normally, no letter in the grid may be used more than once in a word (see **Advanced Play** for exceptions).

Playing the Game

When invoked, boggletool displays a grid of letters and an hourglass. To enter words, simply type in lower case letters to spell the word you want. Use any whitespace (SPACE, TAB, or NEWLINE) to finish a word. To correct mistakes you make, use BACKSPACE or DEL to delete the last character, or use CTRL-U to delete an entire word. **boggletool** verifies that words you enter are both in the grid and are valid English words. If you type in a character which would form a word which is not in the grid, the display will flash and the character you typed will not be echoed. When you type any whitespace to end the current word, **boggletool** will verify that the word is three or more letters long and that it appears in the dictionary. If the word you typed is illegal for either reason, the display will flash and you will have to either erase the word or change it. If you try to enter a valid word which you have already entered, the display will flash and the previous occurrence of the word will be highlighted. Again, you will have to erase the word before continuing. As you enter words, the “sand” in the hourglass will fall. At the end of the time limit, the display will flash and you will no longer be allowed to enter words. After a moment, the computer will display two lists of words: the words you found, and other words which also appear in the grid. To play another game, just type any capital letter (or use the pop-up menu).

Using the Menu

The pop-up menu is invoked by pressing the RIGHT mouse button. There are four items in it, and they work as follows.

Restart Game
Create a new boggletool a new board, reset the timer, and allow you to start from scratch.

Restart Timer
Allows you to cheat by reseting the hourglass timer to zero.

Give Up
End the game and print the results immediately.

Quit Allows you to quit running the boggletool program. A prompt appears asking you to confirm the quit; when it does, click the LEFT mouse button to quit or the RIGHT mouse button to abort the quit.

Advanced Play

There are two options for advanced players. If a single + appears on the command line, letters in the grid may be reused. If two +'s are on the command line, letters may also be considered adjacent to themselves as well as to their neighbors. Although it is far easier to find words with these two options, there are also many more possible words in the grid and it is therefore difficult to find them all.

FILES

/usr/games/boggledict	dictionary file for computer's words

NAME

brotcube – rotate a simple cube

SYNOPSIS

/usr/demo/brotcube

DESCRIPTION

brotcube rotates a skeletal outline of a cube consisting of 14 vectors. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display. Each rotation consists of 100 views.

This program gives an indication of the performance of the SunCore Graphics Package.

Type **q** to exit the program.

NAME

bsuncube – view 3-D Sun logo

SYNOPSIS

/usr/demo/bsuncube

DESCRIPTION

bsuncube allows the user to view a cube from various positions with hidden faces removed. The faces of the cube consist of the Sun logo. The viewing position is selected using the mouse. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display.

The program operates in two modes: **DisplayObject** mode and **SelectView** mode. The program starts in **DisplayObject** mode:

> **DisplayObject**: The cube is displayed in 3-D perspective with hidden faces removed. Type **q** while in this mode to exit the program. Press RIGHT mouse button to switch to **SelectView** mode.
>
> **SelectView**: Schematic projections of the outline of the cube are shown and the mouse is used to select a viewing position. Use LEFT mouse button to set *x* and MIDDLE mouse button to set *y* in the *Front View*. Use MIDDLE mouse button to set *z* in the *Top View*. Press RIGHT mouse button to switch to **DisplayObject** mode.

The view shown in **DisplayObject** mode is drawn using the conventions that the viewer is always looking from the viewing position toward the center of the cube and that the positive *y* axis on the screen is the projection of the positive *y* axis in 3-D cube coordinates.

NAME

buttontest – demonstration and testing program for SunButtons

SYNOPSIS

/usr/demo/BUTTONBOX/buttontest

DESCRIPTION

buttontest displays a window with thirty two buttons, corresponding to those on SunButtons. To determine if the button box has been set up correctly, select the **Diagnostic** button on the panel. If the button box is correctly interfaced, **buttonbox OK** is displayed, and pressing a button on the box highlights a button on the screen. If **No Response from Buttonbox** is displayed, repeat the button box install procedure.

NAME

canfield, canfieldtool, cfscores – Canfield solitaire card game

SYNOPSIS

/usr/games/canfield [**–ac**]

/usr/games/canfieldtool [**–ac**]

/usr/games/cfscores [**–ac**] [*username*]

AVAILABILITY

These games are available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

canfield can be played on any terminal. **canfieldtool** is the SunView version with attractive graphics.

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In **canfield**, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types **ht** for his move. Foundation base cards are also automatically moved to the foundation when they become available.

Canfieldtool

Once you understand the rules, **canfieldtool** is self-explanatory.

Canfield

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (The initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

OPTIONS

a	Print out **canfield** accounts for all users that have played the game since the database was set up.
c	Maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase the chances of winning.

With no arguments, **cfscores** prints out the current status of your canfield account. If *username* is specified, it prints out the status of their account.

FILES

/usr/games/canfield	the game itself
/usr/games/lib/cfscores	the database of scores

BUGS

It is impossible to cheat.

NAME

cdplayer – CD-ROM audio demo program

SYNOPSIS

cdplayer [-d *device*] [*sunview options*]

AVAILABILITY

This demo is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

cdplayer demonstrate the CD quality audio capability of the CD-ROM drive. It is a SunView program and plays any Audio Compact Discs. There are four panels in the window. The top panel displays the all the available tracks on the CD. The user can select the any tracks by clicking it with the left mouse button. The second panel contains the play, pause, stop and eject button. The third panel display the CD music address and track number. The bottom panel contains the volume control slider and close button.

Refer to the CD-ROM hardware documentation for connecting the speakers or head-phones to the drive.

OPTIONS

–d *device*	Use *device* as the CD-ROM device, rather than **/dev/rsr0** the default CD-ROM device.

FILES

/dev/rsr0	CD-ROM raw file

SEE ALSO

sr(4)

NAME

chess – the game of chess

SYNOPSIS

/usr/games/chess

AVAILABILITY

This game is available for Sun-3 and Sun-4 systems with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

chess is a computer program that plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '**+**' is used to specify check; '**o-o**' and '**o-o-o**' specify castling. To play black, type '**first**'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

DIAGNOSTICS

The most cryptic diagnostic is '**eh?**' which means that the input was syntactically incorrect.

FILES

/usr/games/lib/chess.book
book of opening moves

BUGS

Pawns may be promoted only to queens.

NAME

chesstool – window-based front-end to chess program

SYNOPSIS

/usr/games/chesstool [*chess_program*]

AVAILABILITY

This game is available for Sun-3 and Sun-4 systems, with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

chesstool is a window-based front-end to the **chess**(6) program. Used without options, **chesstool** uses **/usr/games/chess**; you can designate any alternate program which uses the same command syntax as **chess**(6) with the *chess_program* argument.

When **chesstool** starts up, it displays a large window with three subwindows. The first subwindow displays messages '**Illegal move**', for example. The second subwindow is an options subwindow; options are described below. The final subwindow is a chessboard display with white and black pieces and two (advisory only) timekeeping clocks.

Make your moves with the mouse: select a piece by positioning the arrow cursor over the piece and pressing the left mouse button down, then drag the piece to the destination square, and release the button. The cursor will then turn to an hourglass icon while the system plays.

Items in the subwindow may be selected with either the left or middle mouse buttons. These options are:

Last Play	Show the last play made.
Undo	Undo your last move and the machine's response. Once the game is over, it is not possible to restart it, so undo will update the board, but the game cannot be continued from that position.
Flash	Flash when the machine has completed its move. When this command is selected, a check mark will appear next to the word **Flash**. In flash mode, if **chesstool** is open, the piece moved by the system on its play will flash until you make your move. If **chesstool** is iconic, the entire icon will flash when the machine has made its move. Thus you can "Close" **chesstool** and be alerted when it's your turn to move. To turn flash mode off, select flash again.
Machine White	Start a new game with the machine playing white.
Human White	Start a new game with the machine playing black.
Quit	Exit from **chesstool**.

There are two moves which are special: castling and capturing a pawn *en*passant. To castle, move the king only. The position of the rook will automatically be updated. Since the king moves two squares when castling, the move is unambiguous. To capture *en*passant, move the pawn to the square occupied by the opposing pawn which will be captured.

SEE ALSO

chess(6)

NAME

ching – the book of changes and other cookies

SYNOPSIS

/usr/games/ching [*hexagram*]

DESCRIPTION

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams*, each symbolized by a particular arrangement of six straight (——) and broken (– –) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each *hexagram* consists of two major sections. The **Judgement** relates specifically to the matter at hand (For instance, "It furthers one to have somewhere to go.") while the **Image** describes the general attributes of the *hexagram* and how they apply to one's own life ("Thus the superior man makes himself strong and untiring.").

When any of the lines has the value six or nine, it is a moving line; for any such line there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second *hexagram* (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow–stalks or tossed coins. The resulting *hexagram* will be the answer to the question.

Using an algorithm suggested by S. C. Johnson, this oracle simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process ID and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin–toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

The impatient modern may prefer to settle for Chinese cookies; try **fortune**(6).

SEE ALSO

It furthers one to see the great man.

DIAGNOSTICS

The great prince issues commands,
Founds states, vests families with fiefs.
Inferior people should not be employed.

BUGS

Waiting in the mud
Brings about the arrival of the enemy.

If one is not extremely careful,
Somebody may come up from behind and strike him.
Misfortune.

NAME

colordemos – demonstrate Sun Color Graphics Display

SYNOPSIS

/usr/demo/cballs
/usr/demo/cdraw
/usr/demo/cphoto *file*
/usr/demo/cpipes
/usr/demo/cshowmap *file*
/usr/demo/csnow
/usr/demo/csuncube
/usr/demo/csunlogo
/usr/demo/cvlsi

DESCRIPTION

colordemos is a collection of simple demonstration programs for the Sun Color Graphics Display. Each program is briefly described below. To exit each program, send an interrupt signal by typing the appropriate key (usually CTRL-C).

cballs	Colliding balls on color display.
cdraw	Draw on the color display (see **draw**(6) for an explanation of how to use **cdraw**).
cphoto *file*	Display dithered color file on color display. Files suitable for display are in **/usr/demo/colorpix.**
cpipes	Colliding pipes on color display.
cshowmap *file*	Display maps. Files suitable for display are in **/usr/demo/segments**.
csnow	Color kaleidoscope.
csuncube	Multicolored Sun logo.
csunlogo	Shaded Sun logo.
cvlsi	Color VLSI layout demo.

FILES

/usr/demo/colorpix
/usr/demo/segments

NAME

craps – the game of craps

SYNOPSIS

/usr/games/craps

DESCRIPTION

craps is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while the user (the *player*) places bets. The player may choose, at any time, to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a ''bankroll'' of $2,000.

The program prompts with:

bet?

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with **bet?** until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The *first* roll is the roll immediately following a bet:

1. On the first roll:

7 or 11	wins for the roller;
2, 3, or 12	wins for the House;
any other number	is the *point*, roll again (Rule 2 applies).

2. On subsequent rolls:

point	roller wins;
7	House wins;
any other number	roll again.

If a player loses the entire bankroll, the House will offer to lend the player an additional $2,000. The program will prompt:

marker?

A **yes** (or **y**) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

Repay marker?

A reply of **yes** (or **y**) indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

How many?

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program will prompt with **How many?** until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the *total* amount of money borrowed will be *automatically* repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

New game?

to give the House a chance to win back its money.

Any reply other than **yes** is considered to be a **no** (except in the case of **bet?** or **How many?**). To exit, send an interrupt (break), DELETE character or CTRL–D The program will indicate whether the player won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

NAME

cribbage – the card game cribbage

SYNOPSIS

/usr/games/cribbage [**−eqr**] *name* . . .

DESCRIPTION

cribbage plays the card game cribbage, with **cribbage** playing one hand and the user the other. **cribbage** initially asks the user if the rules of the game are needed – if so, **cribbage** displays the appropriate section from *According to Hoyle* with **more**(1).

OPTIONS

−e Provide an explanation of the correct score when the player makes mistakes scoring his hand or crib. This is especially useful for beginning players.

−q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

−r Instead of asking the player to cut the deck, **cribbage** will randomly cut the deck.

PLAYING CRIBBAGE

cribbage first asks the player whether he wishes to play a short game (“once around”, to 61) or a long game (“twice around”, to 121). A response of ‘s’ results in a short game, any other response plays a long game.

At the start of the first game, **cribbage** asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, **cribbage** first prints the player’s hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, **cribbage** cuts the deck (if it is the player’s crib) or asks the player to cut the deck (if it’s its crib); in the latter case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn’t have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. **cribbage** keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. **cribbage** requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

SPECIFYING CARDS

Cards are specified as *rank* followed by *suit*. The *ranks* may be specified as one of **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **t**, **j**, **q**, and **k**, or alternatively, one of **ace**, **two**, **three**, **four**, **five**, **six**, **seven**, **eight**, **nine**, **ten**, **jack**, **queen**, and **king**. *Suits* may be specified as **s**, **h**, **d**, and **c**, or alternatively as **spades**, **hearts**, **diamonds**, and **clubs**. A card may be specified as *rank suit*, or *rank* of *suit*. If the single letter *rank* and *suit* designations are used, the space separating the *suit* and *rank* may be left out. Also, if only one card of the desired *rank* is playable, typing the *rank* is sufficient. For example, if your hand was **2h**, **4d**, **5c**, **6h**, **jc**, **kd** and you wanted to discard the king of diamonds, you could type any of **k**, **king**, **kd**, **k d**, **k of d**, **king d**, **king of d**, **k diamonds**, **k of diamonds**, **king diamonds**, or **king of diamonds**,

FILES

/usr/games/cribbage

SEE ALSO

more(1)

NAME

dialtest – demonstration and testing program for SunDials

SYNOPSIS

/usr/demo/DIALBOX/dialtest

DESCRIPTION

dialtest displays a window with eight dials, corresponding to those on SunDials. To determine if the dialbox has been set up correctly, select the **Diagnostic** button on the panel. If the dialbox is correctly interfaced, **Dialbox OK** is displayed, and turning a dial on the box turn a dial on the screen. If **No Response from Dialbox** is displayed, repeat the dialbox install procedure.

NAME

draw, bdraw, cdraw – interactive graphics drawing

SYNOPSIS

/usr/demo/bdraw
/usr/demo/cdraw

DESCRIPTION

The *draw* programs are menu-driven programs which use the mouse, keyboard, bitmap display and optionally the color display to draw objects, drag them around, save them on disk, and so on. **bdraw** is the draw program for the black and white display and **cdraw** is the program for driving the color display.

The main menu items are selected by moving the mouse cursor and pressing the left mouse button. To redraw the display, point at the left edge of the main menu box and press the left button. The main menu items are:

New Seg xlate
Open a new translatable segment. A segment is a collection of attributes and primitives (lines, text, polygons, etc.). A translatable segment may subsequently be positioned.

New Seg xform
Open a new transformable segment. A transformable segment may subsequently be rotated, scaled, or positioned.

Delete Seg To delete a segment, point at any primitive in the segment and press the left button.

Lines To add line primitives to the currently open segment, position cursor, press the left button, ... press right button to quit.

Polygon To add a polygon primitive to the currently open segment, position the cursor, press the left button, . . . press the right button to terminate the boundary definition. Polygons are filled with the current fill attribute.

Raster To add a raster primitive to the currently open segment, position the cursor, press the left button to reposition the box, adjust the box by moving the mouse, press the right button to create the raster primitive comprising the boxed bitmap. A 'rasterfile' is also created on disk for hardcopy purposes (see */usr/include/rasterfile.h*). This 'rasterfile' file may be spooled to a Versatec printer/plotter for hardcopy after exiting from the draw program. The command to do this is **lpr –v rasterfile**.

Text To add a text primitive to the currently open segment, position cursor, press left button, type the text string at the keyboard (back space works), hit return. Text is drawn with the current text attributes.

Marker To add marker primitives to the currently open segment, position cursor, press the left button to place marker, ... press the right button to quit.

Position To position a segment, point at any primitive in the segment, press left button, position the segment, press right button to quit.

Rotate To rotate a transformable segment, point at any primitive in the segment, press left button, move mouse to rotate, press right button to quit.

Scale To scale a transformable segment, point at any primitive in the segment, press the left button, move mouse to scale in x or y, press right button to quit.

Attributes This item brings up the attribute menu. To select an attribute such as text font, region fill texture (color), linestyle, or line width, point at the item and press the left button. Point at the left edge of the menu box to quit.

Save Seg To save a segment on a disk file, point at the segment, press the left button, type the disk file name, hit return.

Restore Seg
To restore a previously saved segment from disk, type file name, hit return.

Exit Exit the draw program.

BUGS

Rasters and raster text do not scale or rotate. If segments completely overlap, only the last one drawn may be picked by pointing with the mouse. This also applies to the menu segments! Therefore, don't cover them up with polygons. If aborted with your interrupt character, you must give the 'reset' command to turn keyboard echo back on and to reset -cbreak. Therefore, use the Exit item in the main menu to exit the program.

NAME

factor, primes – factor a number, generate large primes

SYNOPSIS

/usr/games/factor [*number*]

/usr/games/primes [*number*]

DESCRIPTION

factor reads lines from its standard input. If it reads a positive number, **factor** will factor the number and print its prime factors, printing each one the proper number of times. **factor** exits when it reads zero, a negative number, or something other than a number. If a *number* is given, **factor** will factor the number, print its prime factors, and exit.

primes reads a number from the standard input and prints all primes larger than the given number and smaller than 2^{32} (about 4.3×10^{9}). If a *number* is given, **primes** will use that number rather than reading one from the standard input.

DIAGNOSTICS

Ouch. Input out of range or for garbage input.

NAME

fish – play ''Go Fish''

SYNOPSIS

/usr/games/fish

DESCRIPTION

fish plays the game of "Go Fish", a children's card game. The object is to accumulate "books" of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the "pool" of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing **a**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **j**, **q**, or **k** when asked. Hitting a RETURN character gives you information about the size of my hand and the pool, and tells you about my books. Saying '**p**' as a first guess puts you into "pro" level; the default is pretty dumb.

NAME

fortune – print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [–] [**–alsw**] [*filename*]

DESCRIPTION

fortune with no arguments prints out a random adage. The flags mean:

–a Choose from either list of adages.

–l Long messages only.

–s Short messages only.

–w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

FILES

/usr/games/lib/fortunes.dat

NAME

gaintool – audio control panel

SYNOPSIS

gaintool

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

gaintool is a SunView demonstration program that controls various characteristics of the SPARCstation 1 audio device, see **audio**(4S). Operations performed by **gaintool** affect all audio programs; for instance, adjusting the **Play Volume** instantly changes the output gain, regardless of which program is playing. **gaintool** also detects audio state changes made by other programs, and updates its display accordingly, keeping **gaintool** in sync with the current device configuration.

gaintool demonstrates an important principle involved in the integration of audio in the desktop environment: by enabling global control of important characteristics, it is not necessary for every application to provide an interface for these parameters. For instance, since audio output may be paused from the control panel, it is not strictly necessary that output applications display a **Pause** button of their own. However, such applications may detect that audio output has been paused, and take appropriate action.

Control Panel

Play Volume
This slider adjusts the output volume. Volume levels between 0 and 100 may be selected, where 0 represents infinite attenuation and 100 is maximum gain.

Record Volume
This slider adjusts the recording gain level in the range 0 to 100.

Monitor Volume
This slider adjusts the monitor gain level in the range 0 to 100. Monitor gain controls the amount of audio input signal that is fed through to the output port. For instance, if an audio source (such as a radio or CD-player) is connected directly to the input port, the input signal may be monitored through either the built-in speaker or the headphone jack.

Output This selector switches the audio output port between the built-in speaker and the external headphone jack.

Pause Play
This button may be used to suspend and resume audio output. If audio output is in progress when **Pause** is clicked, it is stopped immediately and subsequent output data remains queued. The button then switches to a **Resume** button that, when clicked, resumes audio output at the point that it was suspended.

If no process has the device open for output when **Pause** is clicked, **gaintool** holds the device open itself, thereby denying other processes output access. Audio programs that simply open and write to the audio device will typically be suspended when they attempt to open the device. Programs that asynchronously poll the device will discover that it is "busy" and may take appropriate action.

Audio Device Status Panel

Pressing the **PROPS** (L3) key brings up a status panel that shows the current state with the its display accordingly, audio applications. Selecting "Done" from the panel menu (or pressing the (L7) key) removes the panel.

Ordinarily, the device status is updated only when a **SIGPOLL** signal is delivered to **gaintool** (see **audio**(4S)). Because of this, the **Active** and **Samples** indicators are not necessarily kept up-to-date. However, when the mouse is positioned over the panel, status is continually updated.

SEE ALSO

audio(4S), **soundtool**(6)

BUGS

Record Volume should be controlled by a separate panel that also provides automatic gain level adjustment capabilities.

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

gammontool – play a game of backgammon

SYNOPSIS

/usr/games/gammontool [*path*]

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

gammontool paints a backgammon board on the screen, and then lets you play against the computer. It must be run in SunWindows. The optional *path* argument specifies an alternate move-generating program, which must be specially designed to run with **gammontool**.

The game has three subwindows: an option window on top, a message window in the middle, and a large board on the bottom. The buttons in the option window are used to restart, double, etc. The message window has two lines: the first tells whose turn it is, and the second displays any errors that occur.

The Initial Roll

To start the game, roll the dice to determine who goes first. Move the mouse arrow onto the board and click the left button. One die appears on each side of the board: the die on the left is yours, and the die on the right is the computer's. If your roll is greater, then you move; if not, the computer makes a move.

Making Your Move

When it is your turn, '**Your**move' appears in the message window. Place the mouse over any piece of your color, and click the left button. While holding down the button, move the mouse to drag the piece; the piece follows the mouse until you release the button. The tool checks each move and does not allow illegal moves. When you have made as many moves as you can, the computer takes its turn; after it finishes, you may either roll again, or double.

Doubling

To double, click the *Double* button in the option window and wait for the computer's response. If the computer doubles you, a message is displayed and you must answer with the **Accept Double** or **Refuse Double** buttons. The **Forfeit** button can also be used to refuse a double. If the game is doubled, a doubling cube with the proper value is displayed on the bar strip. If the number is facing up, then you may double next. If the number is upside down, it is the computer's turn to double.

Other Buttons

If you want to change your move before you have finished it, use the **Redo Move** or **Redo Entire Move** buttons in the option window. **Redo Entire Move** replaces all of the pieces you have moved so that you can redo them all. **Redo Move** only replaces the last piece you moved, so it is useful when you roll doubles and want to redo only the last piece you moved. Note that once you have made all of the moves your roll permits, play passes immediately to the computer, so you cannot redo the very last move. The **Show Last Move** button allows you to see the last move again.

Leaving the Game

If you want to quit playing backgammon, use the **Quit** button. If you want to forfeit the game, use the **Forfeit** button. The computer penalizes you by taking a certain number of points, but the program does not terminate.

To play another game after winning, losing, or forfeiting, click the **New Game** button. To change the color of your pieces, click the mouse button while pointing at either the **White** or **Black** checkboxes. You may change colors at any time, even in the middle of a game. Changing colors in the middle of a game does not mean that you trade places with the computer; your pieces stay where they are, but they are repainted with the new color. Your pieces always move from the top right to the bottom right of the board, regardless of your color. As an additional cue as to your color, your dice are always displayed on the left half of the board.

Log File

If a there is a **gammonlog** file your home directory, **gammontool** keeps a log of the games played. Each move and double gets recorded, along with the winners and accumulated scores.

FILES

~/gammonlog	log of games played
/usr/games/lib/gammonscores	log of wins and losses

BUGS

The default strategy used by the computer is very poor.

If a single move uses more than one die (for instance if you roll 5, 6 and move 11 spaces without touching down in the middle) it is unpredictable where the program will make the piece touch down. This may be important if there is a blot on one of these middle points. The program will always make the move if possible, but if two midpoints would work and there is a blot on one of them, it is much better to explicitly hit the blot and then move the piece the rest of the way.

NAME

gp_demos, flight, rotobj – demonstration programs for the Graphics Processor

SYNOPSIS

/usr/demo/flight

/usr/demo/rotobj [*object*]

AVAILABILITY

These demos are available with the *Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

These demos only run in windows running on a Graphics Processor surface.

Flight

flight is a mouse-driven flight simulator.

Interactive Commands

Middle-Button	Restart the program.
Right-Button	Increase speed.
Left-Button	Decrease speed.
Move-Mouse-Forward	The airplane dives.
Move-Mouse-Backward	The airplane climbs.
Move-Mouse-Left/Right	The airplane banks.
Left/Right-With-Right-Button	The airplane rolls without banking.

Rotobj

rotobj rotates an *object*. Object files are located in **/usr/demo/DATA** and have the suffix **.vecs**.

FILES

/usr/demoDATA

SEE ALSO

graphics_demos(6)

NAME

graphics_demos, bouncedemo, framedemo, jumpdemo, spheresdemo, – graphics demonstration programs

SYNOPSIS

/usr/demo/bouncedemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

/usr/demo/framedemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

/usr/demo/jumpdemo [**–c**] [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

/usr/demo/spheresdemo [**–d** *dev*] [**–n***x*] [**–r**] [**–q**]

AVAILABILITY

These demos are available with the *Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

bouncedemo

bouncedemo displays a bouncing square.

framedemo

framedemo

displays a series of frames, each of which contains a 256 by 256 image one-bit-deep pixels (that is, the image is a square monochrome bitmap, with 256 bits on a side). **framedemo** looks for the frames in the files **frame.1** through **frame.n** in the current working directory, and displays them in numerical order. A set of sample frames is available in the directory **/usr/demo/globeframes/***.

Interactive Commands

If you move the cursor onto the image surface, you can type certain commands to affect the rate at which the frames are displayed. The initial rate is one frame per second:

f	Remove 1/20th of a second from the interval.
F	Remove one second from the interval. **Ff** makes the interval as small as possible.
s	Add 1/20th of a second.
S	Add one second.

jumpdemo

jumpdemo simulates the famous **Star Wars** jump to light-speed-sequence using vector drawing. Colored stars are drawn on color surfaces.

spheresdemo

spheresdemo computes a random collection of shaded spheres. Colored spheres are drawn on color surfaces.

OPTIONS

–c Rotate the color map to produce a sparkling effect.

–d *surface*
Run the demo on a surface other than the window or system console, for instance:

```
bouncedemo –d /dev/cgone0
```

–n*x* Draw *x* items, or repeat a sequence *x* times.

–r Retain the window. This allows the image to reappear when uncovered instead of restarting the demo.

–q Quick exit. Useful for running several demos from within a shell script.

NAME

hack – replacement for rogue

SYNOPSIS

hack [**–d** *hackdir*] [**–s all** | *player* . . .]

DESCRIPTION

hack is a display-oriented dungeons & dragons type game. Both display and command structure resemble *rogue*, although **hack** has twice as many monster types and requires three times as much memory.

Normally **hack** looks in **/usr/games/lib/hackdir** for the files listed below; this directory can be changed with the **–d** option. The **–s** option permits you to search the player record. Given the keyword **all**, **hack** lists all players; given the login name of a player, it lists all scores of that player.

FILES

record	top 100 list (start with an empty file)
news	changes or bugs (start with no news file)
data	information about objects and monsters
help	introductory information (no doubt outdated)
hh	compacted version of help
perm	empty file used for locking
rumors	texts for fortune cookies

NAME

hangman – computer version of the game hangman

SYNOPSIS

/usr/games/hangman

DESCRIPTION

In **hangman,** the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

FILES

/usr/dict/words	on-line word list

NAME

hunt – a multiplayer multiterminal game

SYNOPSIS

/usr/games/hunt[**–m**] [*hostname*] [**–l** *name*]

DESCRIPTION

The object of the game **hunt** is to kill off the other players. There are no rooms, no treasures, and no monsters. Instead, you wander around a maze, find grenades, trip mines, and shoot down walls and players.

Your score is the ratio of number of kills to number of times you entered the game and is only kept for the duration of a single session of **hunt**. The more players you kill before you die, the better your score is.

hunt normally looks for an active game on the local network; if none is found, it starts one up on the local host. One may specify the location of the game by giving the *hostname*argument.

hunt only works on crt (vdt) terminals with at least 24 lines, 80 columns, and cursor addressing. The screen is divided in to 3 areas. On the right hand side is the status area. It shows you how much damage you've sustained, how many charges you have left, who's in the game, who's scanning (the asterisk in front of the name), who's cloaked (the plus sign in front of the name), and other players' scores. Most of the rest of the screen is taken up by your map of the maze, except for the 24th line, which is used for longer messages that do not fit in the status area.

hunt uses the same keys to move as **vi** does, for instance, **h**,**j**,**k**, and **l** for left, down, up, right respectively. To change which direction you're facing in the maze, use the upper case version of the movement key (for instance, HJKL).

Other commands are:

f	Fire (in the direction you're facing) (Takes 1 charge)
g	Throw grenade (in the direction you're facing) (Takes 9 charges)
F	Throw satchel charge (Takes 25 charges)
G	Throw bomb (Takes 49 charges)
o	Throw small slime bomb (Takes 15 charges)
O	Throw big slime bomb (Takes 30 charges)
s	Scan (where other players are) (Takes 1 charge)
c	Cloak (where you are) (Takes 1 charge)
^L	Redraw screen
q	Quit

Knowing what the symbols on the screen often helps:

–|+	Walls
/\hl288u+288uDiagonal (deflecting) walls	
#	Doors (dispersion walls)
;	Small mine
g	Large mine
:	Shot
o	Grenade
O	Satchel charge
@	Bomb
s	Small slime bomb
$	Big slime bomb
><^v	You facing right, left, up, or down

} { i !	Other players facing right, left, up, or down
*	Explosion
\\\|/ -*E- /|\\	Grenade and large mine explosion

Satchel and bomb explosions are larger than grenades (5x5, 7x7, and 3x3 respectively).

Other helpful hints:

You can only fire in the direction you are facing.
You can only fire three shots in a row, then the gun must cool.
A shot only affects the square it hits.
Shots and grenades move 5 times faster than you do.
To stab someone,
    you must face that player and move at them.
Stabbing does 3 points worth of damage and shooting does 5 points.
You start with 15 charges and get 5 more for every new player.
A grenade affects the nine squares centered about the square it hits.
A satchel affects the twenty-five squares centered about the square it hits.
A bomb affects the forty-nine squares centered about the square it hits.
One small mine and one large mine is placed in the maze for every new player.
A mine has a 5% probability of tripping when you walk directly at it;
    50% when going sideways on to it; 95% when backing up on to it.
Tripping a mine costs you 5 points or 10 points respectively.
Defusing a mine is worth 1 charge or 9 charges respectively.
You cannot see behind you.
Scanning lasts for (20 times the number of players) turns.
    Scanning takes 1 ammo charge, so do not waste all your charges scanning.
You get 2 more damage capacity points and 2 damage points taken away
    whenever you kill someone.
Maximum typeahead is 5 characters.
A shot destroys normal (for instance, non-diagonal, non-door) walls.
Diagonal walls deflect shots and change orientation.
Doors disperse shots in random directions (up, down, left, right).
Diagonal walls and doors cannot be destroyed by direct shots but may
    be destroyed by an adjacent grenade explosion.
Walls regenerate, reappearing in the order they were destroyed.
    One percent of the regenerated walls will be diagonal walls or doors. When a wall is generated directly beneath a player, he is thrown in a random direction for a random period of time. When he lands, he sustains damage (up to 20 percent of the amount of damage he had before impact); that is, the less damage he had, the more nimble he is and therefore less likely to hurt himself on landing.

ENVIRONMENT

The environment variable HUNT is checked to get the player name. If you do not have this variable set, **hunt** will ask you what name you want to play under. You may also set up a single character keyboard map, but then you have to enumerate the options. For example:

setenv HUNT "name=Sneaky,mapkey=zoFfGg1f2g3F4G"

sets the player name to Sneaky, and the maps **z** to **o**, **F** to **f**, **G** to **g**, **1** to **f**, **2** to **g**, **3** to **F**, and **4** to **G**.

The *mapkey* option must be last.

It is a boring game if you are the only one playing.

OPTIONS

−m You enter the game as a monitor (you can see the action but you cannot play).

−l *name* Enter the game as player *name*.

FILES

/usr/games/lib/hunt.driver game coordinator

LIMITATIONS

hunt normally drives up the load average to be about (number_of_players + 0.5) greater than it would be without a **hunt** game executing. A limit of three players per host and nine players total is enforced by **hunt**.

BUGS

To keep up the pace, not everything is as realistic as possible.

NAME

life – John Conway's game of life

SYNOPSIS

/usr/games/life

AVAILABILITY

This game is available with the *Games* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

life is a program that plays John Conway's game of life. It only runs under **sunview**(1).

When invoked, **life** will display a window with a small control panel at the top, and a large drawing area at the bottom. You can create pieces in the drawing area with the left button, and erase them with the middle button. When you select **Run** in the control panel, the pieces will begin to evolve, and the drawing region will update itself at a speed controlled by the slider labeled with **Fast** and **Slow**. **life** keeps track of all the pieces even if they are not visible. The scroll bars surrounding the drawing region can be used to see pieces that have moved out of view. There are some standard patterns that can be drawn by popping up a menu in the drawing subwindow.

The meaning of the items in the first row of the control panel (from left to right) are as follows. If you click on the picture which looks like a tic-tac-toe board, a grid will appear in the drawing region. If you click on **Step**, the mode will change from run mode (where the pieces update continuously) to step mode (where an update is only done when you click on **Step**). Following **Gen** is a number indicating the number of generations that have occurred. The button marked **Find** will scroll so that at least one piece is in view. This is useful when all the pieces disappear from view. The button marked **Clear** will clear the drawing region, but leave the other controls unchanged. **Reset** will reset all the panel controls, but will not erase any of the pieces, and **Quit** Exits the tool. The second row contains two sliders. The first controls the update speed when in run mode, the second controls the size of the pieces.

SEE ALSO

sunview(1)

NAME

mille – play Mille Bornes

SYNOPSIS

/usr/games/mille [file]

DESCRIPTION

mille plays a two-handed game reminiscent of the Parker Brother's game of Mille Bornes with you. The rules are described below. If a file name is given on the command line, the game saved in that file is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P	Pick a card from the deck. This card is placed in the 'P' slot in your hand.
D	Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a carriage-return or space. The carriage-return or space is required to allow recovery from typos which can be very expensive, like discarding safeties.
U	Use a card. The card is again indicated by its number, followed by a carriage-return or space.
O	Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.
Q	Quit the game. This will ask for confirmation, just to be sure. Hitting DELETE (or RUBOUT) is equivalent.
S	Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a RETURN character without a name, the save will be terminated and the game resumed.
R	Redraw the screen from scratch. The command ^L (CTRL–L) will also work.
W	Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "S" command to reattempt the save. (The game itself is a product of Parker Brothers, Inc.)

SEE ALSO

curses(3V)

CARDS

Here is some useful information. The number in brackets after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas [2]	Gasoline [6]	Extra Tank [1]
Flat Tire [2]	Spare Tire [6]	Puncture Proof [1]
Accident [2]	Repairs [6]	Driving Ace [1]
Stop [4]	Go [14]	Right of Way [1]
Speed Limit [3]	End of Limit [6]	

25 – [10], 50 – [10], 75 – [10], 100 – [12], 200 – [4]

RULES

Object: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. With the exception of the *speed limit* card, they can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA (unlabeled)**: This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaches 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastiness.

Go (Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).

Stop is played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.

Speed Limit is played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.

End of Limit is played on your Speed pile to nullify a *Speed Limit* played by your opponent.

Out of Gas is played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.

Flat Tire is played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.

Accident is played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn.*

Right of Way prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.

Extra Tank When played, your opponent cannot play an *Out of Gas* on your Battle Pile.

Puncture Proof When played, your opponent cannot play a *Flat Tire* on your Battle Pile.

Driving Ace When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand.* A hand ends whenever one player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action.*

Coup Fouré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bornes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played: Each player scores as many miles as they played before the trip ended.
Each Safety: 100 points for each safety in the Safety area.
All 4 Safeties: 300 points if all four safeties are played.
Each Coup Fouré: 300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed: 400 points bonus for completing the trip to 700 or 1000.
Safe Trip: 300 points bonus for completing the trip without using any 200 mile cards.
Delayed Action: 300 points bonus for finishing after the deck was exhausted.
Extension: 200 points bonus for completing a 1000 mile trip.
Shut-Out: 500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

NAME

monop – Monopoly game

SYNOPSIS

/usr/games/monop [*filename*]

DESCRIPTION

monop is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must "fix the problem", that is, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, for instance a name, place or person, you can type **?** to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

USAGE

Commands

quit: Quit game. This allows you to quit the game. It asks you if you are sure.

print Print board. This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

Name	The first ten characters of the name of the square
Own	The *number* of the owner of the property.
Price	The cost of the property (if any)
Mg	This field has a '*' in it if the property is mortgaged
#	If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
Rent	Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings :
List your own holdings, that is, money, get-out-of-jail-free cards, and property.

holdings:
Holdings list. Look at anyone's holdings. It will ask you whose holdings you wish to look at. When you are finished, type **done**.

shell: Shell escape. Escape to a shell. When the shell dies, the program continues where you left off.

mortgage:
Mortgage property. Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage:
Unmortgage property. Unmortgage mortgaged property.

buy: Buy houses. Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: Sell houses. Sets up a list of monopolies from which you can sell houses. it operates in an analogous manner to **buy**

card: Card for jail. Use a get-out-of-jail-free card to get out of jail. If you are not in jail, or you do not have one, it tells you so.

pay: Pay for jail. Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you are not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: Save game. Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the **monop** command, or by using the **restore** command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore:
Restore game. Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the RETURN key instead of a command, it is the same as typing *roll*.

FILES

/usr/games/lib/cards.pck	chance and community chest cards

BUGS

No command can be given an argument instead of a response to a query.

NAME

moo – guessing game

SYNOPSIS

/usr/games/moo

DESCRIPTION

moo is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A ''cow'' is a correct digit in an incorrect position. A ''bull'' is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).

NAME

number – convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

number copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

play – play audio files

SYNOPSIS

play [**–i**] [**–V**] [**–d** *dev*] [**–v** *vol*] [*filename* . . .]

AVAILABILITY

This command is only available with the *Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

play copies the named audio files to the audio device. Audio files named on the command line are played sequentially. If no filenames are present, the standard input stream is played. The special filename '–' may be used to read the standard input stream instead of a file.

The input files (including the standard input) must contain a valid audio file header. The encoding information in this header is matched against the capabilities of the audio device and, if the data formats are incompatible, an error message is printed and the file is skipped.

Minor deviations in sampling frequency (those less than 1%) are ordinarily ignored. This allows, for instance, data sampled at 8012 Hz to be played on an audio device that only supports 8000 Hz. If the –V option is specified, such deviations are flagged with warning messages.

OPTIONS

–i	Print an error message and exit immediately if the audio device is unavailable (that is, another process currently has write access). **play** will ordinarily wait until it can obtain access to the device.
–V	Verbose. Print messages to the standard error while waiting for access to the audio device or when sample rate deviations are detected.
–d *dev*	Specify an alternate audio device to which output should be directed. If the **–d** option is not specified, **/dev/audio** is the default audio device.
–v *vol*	Set the output volume to *vol* before playing begins. *vol* is an integer value between 0 and 100, inclusive. If this argument is not specified, the output volume remains at the level most recently set by any process.
–?	Help. Print a command line usage message.

SEE ALSO

record(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

primes – print all primes larger than some given number

SYNOPSIS

/usr/games/primes [*number*]

DESCRIPTION

primes reads a number from the standard input and prints all primes larger than the given number. If *number* is given as an argument, it uses that number rather than reading one from the standard input.

BUGS

It obviously cannot print *all* primes larger than some given number. It will not behave very sensibly when it overflows an **int**.

NAME

quiz – test your knowledge

SYNOPSIS

/usr/games/quiz [**–i***filename*] [**–t**] [*category1 category2*]

DESCRIPTION

quiz gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, **quiz** gives instructions and lists the available categories.

quiz tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, **quiz** reports a score and terminates.

The **–t** flag specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **–i** flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category newline | category ':' line
category = alternate | category '|' alternate
alternate = empty | alternate primary
primary  = character | '[' category ']' | option
option    = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with **sh**(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, **quiz** will refrain from asking it.

FILES

/usr/games/quiz.k/*

BUGS

The construct 'a | ab' doesn't work in an information file. Use 'a{b}'.

NAME

rain – animated raindrops display

SYNOPSIS

/usr/games/rain

DESCRIPTION

rain's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use **termcap**, the TERM environment variable must be set (and exported) to the type of the terminal being used.

FILES

/etc/termcap

NAME

random – select lines randomly from a file

SYNOPSIS

/usr/games/random [**–er**] [*divisor*]

DESCRIPTION

random acts as a text filter, randomly selecting lines from its standard input to write to the standard output. The probability that a given line is selected is normally 1/2; if a *divisor* is specified, it is treated as a floating-point number, and the probability is 1/*divisor* instead.

OPTIONS

–e　Don't read the standard input or write to the standard output. Instead, exit with a random exit status between 0 and 1, or between 0 and *divisor*-1 if *divisor* is specified.

–r　Don't buffer the output. If **–r** is not used, output is buffered in blocks, or line-buffered if the standard output is a terminal.

NAME

raw2audio – convert raw audio data to audio file format

SYNOPSIS

raw2audio [**–f**] [**–c** *chan*] [**–e** *enc*] [**–i** *info*] [**–o** *cnt*] [**–p** *bits*] [**–s** *rate*] [*filename* ...]

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

raw2audio adds an audio file header to the named raw data files. The encoding information in this header is taken from the command line options.

If no filenames are specified, **raw2audio** reads raw data from the standard input stream and writes an audio file to the standard output. If a target file is a symbolic link, the underlying file will be rewritten.

OPTIONS

–f	Force. If an input file already contains an audio file header, **raw2audio** ordinarily prints a warning message and skips the file. If the **–f** flag is specified, the old file header, including the 'information' field, is replaced.
–c *chan*	Specify the number of interleaved audio channels in each sample frame. If not specified, a single channel is assumed.
–e *enc*	Specify the encoding type. *enc* may be one of the following: **ULAW**, **LINEAR**, or **FLOAT**, corresponding to µ-law, integer PCM, and IEEE floating-point formats, respectively. If not specified, µ-law encoding is assumed.
–i *info*	Specify the 'information' field of the output file header.
–o *cnt*	Specify the number of bytes to skip in the audio data stream. This option may be used, for instance, to extract audio data from files containing unrecognizable file headers.
–s *rate*	Specify the sample rate frequency, in Hz. If not specified, the sample rate defaults to 8000 Hz.
–p *bits*	Specify the sound unit size, in bits. If not specified, the precision defaults to 8 bits.
–?	Help. Print a command line usage message.

SEE ALSO

play(6), **record**(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

record – record an audio file

SYNOPSIS

record [**–a**] [**–f**] [**–d** *dev*] [**–i** *info*] [**–t** *time*] [**–v** *vol*] [*filename*]

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

record copies audio data from the audio device to a named audio file. The output file will be prefixed by an audio file header. The encoding information in this header is taken from the configuration of the audio device. If no filename is present, or if the special filename '–' is specified, output is directed to the standard output stream.

Recording begins immediately and continues until a **SIGINT** signal (CTRL-C) is received. If the **–t** option is specified, *record* stops when the specified quantity of data has been recorded.

If the audio device is unavailable (that is, another process currently has read access), **record** prints an error message and exits immediately.

OPTIONS

–a	Append the data on the end of the named audio file. The audio encoding of the file must match the audio device configuration.
–f	Force. When the **–a** flag is specified, the sample rate of the audio file must match the device configuration. If the **–f** flag is also specified, sample rate differences are ignored, with a warning message printed to the standard error.
–d *dev*	Specify an alternate audio device from which input should be taken. If the **–d** option is not specified, **/dev/audio** is used as the default audio device.
–i *info*	The 'information' field of the output file header is set to the string specified by the *info* argument. This option may not be specified in conjunction with the **–a** argument.
–t *time*	The *time* argument specifies the maximum length of time to record. Time may be specified as a floating-point value, indicating the number of seconds, or in the form: *hh***:***mm***:***ss***.dd**, where hour and minute specifications are optional.
–v *vol*	Specify the recording gain. *vol* is an integer value between 0 and 100, inclusive. If this argument is not specified, the input volume will remain at the level most recently set by any process.
–?	*Help*: Print a command line usage message.

SEE ALSO

play(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

robots – fight off villainous robots

SYNOPSIS

/usr/games/robots [**–sjta**] [*scorefile*]

DESCRIPTION

robots pits you against evil robots, who are trying to kill you (which is why they are evil). Fortunately for you, even though they are evil, they are not very bright and have a habit of bumping into each other, thus destroying themselves. In order to survive, you must get them to kill each other off, since you have no offensive weaponry.

Since you are stuck without offensive weaponry, you are endowed with one piece of defensive weaponry: a teleportation device. When two robots run into each other or a junk pile, they die. If a robot runs into you, you die. When a robot dies, you get 10 points, and when all the robots die, you start on the next field. This keeps up until they finally get you.

Robots are represented on the screen by a '+', the junk heaps from their collisions by a '*', and you (the good guy) by a '@'.

The commands are:

h	move one square left
l	move one square right
k	move one square up
j	move one square down
y	move one square up and left
u	move one square up and right
b	move one square down and left
n	move one square down and right
.	(also space) do nothing for one turn
HJKLBNYU	run as far as possible in the given direction
>	do nothing for as long as possible
t	teleport to a random location
w	wait until you die or they all do
q	quit
^L	redraw the screen

All commands can be preceded by a count.

If you use the 'w' command and survive to the next level, you will get a bonus of 10% for each robot which died after you decided to wait. If you die, however, you get nothing. For all other commands, the program will save you from typos by stopping short of being eaten. However, with 'w' you take the risk of dying by miscalculation.

Only five scores are allowed per user on the score file. If you make it into the score file, you will be shown the list at the end of the game. If an alternate score file is specified, that will be used instead of the standard file for scores.

OPTIONS

–s Do not play, just show the score file.

–j Jump, when you run, don't show any intermediate positions; only show things at the end. This is useful on slow terminals.

−t Teleport automatically when you have no other option. This is a little disconcerting until you get used to it, and then it is very nice.

−a Advance into the higher levels directly, skipping the lower, easier levels.

FILES

/usr/games/lib/robots_roll	the score file

BUGS

Bugs? You *crazy*, man?!?

NAME

rotcvph – rotate convex polyhedron

SYNOPSIS

/usr/demo/rotcvph*filename*

DESCRIPTION

rotcvph rotates a convex polyhedron with hidden surfaces removed. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display. The mandatory file argument contains a polygonal object definition as described below.

Initially the program displays a fixed view of the object. The following commands may be typed at any time:

n	Display successive views with no waiting.
w	Wait for SPACE to be typed before displaying each view.
q	Exit the program.

The format of the polygonal object definition is illustrated by this example of the definition of a pyramid:

```
          5       5
-1.0 1.0 -1.0 1.0 -1.0 1.0
1.0  1.0 -1.0
1.0 -1.0 -1.0
-1.0 -1.0 -1.0
-1.0  1.0 -1.0
0.0  0.0  1.0
4         4 3 2 1
3         1 5 4
3         2 5 1
3         3 5 2
3         4 5 3
```

The first line gives the number of vertices followed by the number of polygons. The second line gives the coordinates of a bounding box for the object. Minimum and maximum coordinate values are given for each of three dimensions in the order *minx*, *maxx*, *miny*, *maxy*, *minz*, *maxz*. Lines 3 through v+2 (where v is the number of vertices) give vertex coordinates in the order *x*, *y*, ,IR z . Lines v+3 through v+p+2 (where p is the number of polygons) give polygon descriptions. The first number is the number of vertices for the polygon. Succeeding numbers on the line are indices into the vertex list. Polygons should be planar. Coordinates are given in floating point format and everything else is integer. Entries on a given line are separated by arbitrary whitespace. A maximum of 400 vertices and 400 polygons may be defined. The polygon definitions may contain a maximum of 1600 instances of the vertices. **/usr/demo/data** contains several object definition files, including **icosa.dat, socbal.dat**, and **pyramid.dat.**

The above format may be used to define non-convex objects. The program will display these objects but hidden surface computations will not be done correctly.

FILES

/usr/demo/data/*.dat	sample object definition files
icosa.dat	
socbal.dat	
pyramid.dat	

BUGS

All floating point transformations are done twice for each view, once to draw the object and once to undraw it.

Lines which are common to two visible polygons in a view are drawn twice, once for each polygon.

NAME

snake, snscore – display chase game

SYNOPSIS

/usr/games/snake [**–w***n*] [**–l***n*]
/usr/games/snscore

DESCRIPTION

snake is a display-based game which must be played on a CRT terminal from among those supported by **vi**(1). The object of the game is to make as much money as possible without getting eaten by the snake. The **–l** and **–w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as **vi**(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc	These keys are like hjkl but form a directed pad around the d key.
HJKL	These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.
SEFC	Likewise for the upper case versions on the left.
ATPB	These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, for example, P is at the far right of the keyboard.
x	This lets you quit the game at any time.
p	Points in a direction you might want to go.
w	Space warp to get out of tight squeezes, at a price.
!	Shell escape
^Z	Suspend the snake game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

To see who wastes time playing snake, run **/usr/games/snscore.**

FILES

/usr/games/lib/snakerawscores	database of personal bests
/usr/games/lib/snake.log	log of games played

BUGS

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen. A perfect function to do this equitably has not been devised.

NAME

soundtool – audio play/record tool

SYNOPSIS

soundtool

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

soundtool is a SunView demonstration program that allows recording, playing, and simple editing of audio data. The display consists of six regions: a play/record control panel, a function control panel, an oscilloscope, a display control panel, a waveform display panel, and a pop-up audio status panel.

Play/Record Control Panel

Play/Stop
Clicking this button plays the currently selected region of data. While data is playing this button becomes a **Stop** button. If audio output is busy when **Play** is started, this button displays **Waiting**. When the device is available, the button switches to **Stop** and audio output begins. Clicking on the **Waiting** button resets the tool to the idle state.

Record/Stop
Clicking this button starts the recording of data from the audio input port that is wired to the 8-pin mini-DIN connector on the back of SPARCstation 1 systems. While recording is in progress, this button becomes a **Stop** button. If audio input is busy when **Record** is selected, an alert pops up and the tool resets to the idle state. A maximum of 5 minutes may be recorded at a time.

Pause Clicking this button while playing or recording suspends the current operation. The button becomes a **Resume** button that may be selected to continue the suspended operation.

Describe
Clicking this button brings up the "Audio Status Panel". If the panel was already visible, clicking this button removes it.

Quit Clicking this button exits **soundtool**.

Play Volume
This slider adjusts the playback volume. Volume levels between 0 and 100 may be selected, where 0 represents infinite attenuation and 100 is maximum gain.

Record Volume
This slider adjusts the recording level in the range 0 to 100.

Output To
This selector switches the audio output port between the built-in speaker and the external headphone jack.

Looping
When **Looping** is disabled, the current data region (that is, the data between the two markers in the waveform display) is played once. If **Looping** is enabled, the selected data plays endlessly until the **Stop** button is pressed.

Function Control Panel

Load Clicking **Load** reads in the audio file specified by the **Directory** and **File** fields. If the named file does not contain a valid audio header, the raw data is copied into the buffer and an alert is displayed. Clicking the **Store** button at that point rewrites the file with the proper audio file header.

Arbitrarily large audio files may be loaded. However, system swap space resources may be depleted (one minute of SPARCstation 1 audio data consumes roughly .5 Mbyte of swap space).

Store Clicking **Store** writes the selected data region into the file specified by the **Directory** and **File** fields. If the named file exists, an alert will request confirmation of the operation.

Append

Clicking **Append** appends the selected data region to the file specified by the **Directory** and **File** fields. The named file must contain a valid audio file header.

Directory

The **Directory** field specifies a directory path in which to look for audio files.

File The **File** field designates the file to be loaded from, stored to, or appended to. Holding down the right mouse button on this field presents a menu of audio files in the currently designated directory. All files that contain a valid audio file header, or whose names have the suffix **.au** or **.snd**, are listed.

Oscilloscope

When the program is in the idle state and the cursor is in the waveform display panel, the oscilloscope acts as a magnifying glass, displaying the region of the audio waveform that is currently under the cursor. When the program is playing or recording, the oscilloscope displays the data that is currently being transferred. Note: there is a small time lag in the display of recorded data, due to the fact that the audio device driver buffers input data and delivers it to the application in discrete segments.

Display Control Panel

Zoom The **Zoom** slider adjusts the compression factor used in the display of the waveform. The upper compression limit is chosen so that the entire waveform fits in the waveform display panel. The lower limit is restricted by the ability to manipulate large scrolling regions in SunView. Adjustment of the **Zoom** slider ordinarily results in data compression or expansion around the center of the currently displayed waveform. If the waveform display contains one or both data selection markers, an attempt is made to keep at least a portion of the selected data region in the window.

The magnified waveform presented in the oscilloscope display is unaffected by the **Zoom** value. However, cursor movement over the waveform reflects the current compression; that is, lower **Zoom** values result in finer granularity of mouse movement.

Waveform Display Panel

The waveform display shows all or part of the current waveform, depending on the current **Zoom** value. Scrolling of the waveform may be achieved either by using the scrollbar or by dragging the waveform to the right or left while holding the middle mouse button down. Note: scrolling is disabled when the entire waveform is being displayed (that is, when the **Zoom** value is at its maximum).

In some cases, it is desirable to identify a subset of the waveform. For instance, the **Play**, **Store**, and **Append** functions operate on a selected region, rather than the entire waveform. The currently selected region of interest is delimited by dashed vertical lines. A new region may be selected by clicking the left or right mouse button and dragging it across the desired region of interest. Alternatively, a single click on the left or right mouse buttons adjusts the start or end points.

Audio Status Panel

This panel is displayed (or removed) when the **Describe** button is pressed. It contains fields that describe the data in the buffer.

Sample Rate

This field displays the sampling frequency, in samples per second.

Channels

This field denotes the number of interleaved channels of audio data.

Precision

This field identifies the encoding precision, in bits per sample.

Encoding
This field displays the encoding format.

Total Length
This field shows the length of the entire data buffer, in the form *hh:mm:ss.dd*.

Selection
This field identifies the start and end times of the currently selected region of interest.

Info String
When an audio file is loaded, the first 80 characters of the information field of the audio header are displayed in this field. This string may be edited, though the new information is only written out when the **Store** operation is performed.

BUGS

Currently, **soundtool** is capable of displaying only 8-bit μ-law encoded data. This restriction should be removed.

Audio files should be mapped in order to reduce the swap space requirements. The limit on recording length should also be removed.

SunView scrollbars operate on canvases whose virtual size is given by a short integer (that is, 16 bits). This ridiculous constraint is the reason for the lower limit on zooming. Because of this, the accuracy of start and end point selection is reduced when the data buffer is large.

Region selections made over the waveform display panel work best when the click and drag paradigm is used. Adjusting the start or end points by a single click is susceptible to error; that is, if the mouse moves slightly between the button down and up events, the result is a very small selection.

SEE ALSO

gaintool(6), **play**(6), **raw2audio**(6), **record**(6)

WARNINGS

This program is furnished on an *as is* basis as a demonstration of audio applications programming.

NAME

suncoredemos – demonstrate SunCore Graphics Package

SYNOPSIS

/usr/demo/cproduct
/usr/demo/cshademo

AVAILABILITY

This command is only available with the *Demos* installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

suncoredemos is a collection of simple programs demonstrating the SunCore Graphics Package. Each program is briefly described below. These programs generate all graphics output using subroutine calls to SunCore. To exit each program, generate an interrupt signal by typing the appropriate key (usually DELETE).

cproduct	Color Sun architecture demo (requires Sun Color Graphics Display).
cshademo	Shaded surface polygons demo (requires Sun Color Graphics Display).

NAME

sunview_demos, canvas_demo, cursor_demo – Window-System demonstration programs

SYNOPSIS

/usr/demo/canvas_demo

/usr/demo/cursor_demo

AVAILABILITY

These demos are available with the *SunView Demos* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

canvas_Demo

canvas_demo demonstrates the capabilities of the canvas subwindow package. It consists of two subwindows: a control panel and a canvas. By adjusting the items on the control panel, you can manipulate the attributes of the canvas, and see the results.

cursor_Demo

cursor_demo demonstrates what you can do with cursors. A single control panel is provide for adjusting the various cursor attributes. As you adjust the items on the control panel, the panel's cursor changes in appearance.

NAME

trek – trekkie game

SYNOPSIS

/usr/games/trek [[–a] *filename*]

DESCRIPTION

trek is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the –a flag is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commodore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

COMMAND SUMMARY

abandon	capture
cloak up/down	
computer request; ...	damages
destruct	dock
help	impulse course distance
lrscan	move course distance
phasers automatic amount	
phasers manual amt1 course1 spread1 ...	
torpedo course [yes] angle/no	
ram course distance	rest time
shell	shields up/down
srscan [yes/no]	
status	**terminate** yes/no
undock	visual course
warp warp_factor	

NAME

vwcvph – view convex polyhedron

SYNOPSIS

/usr/demo/vwcvph *filename*

DESCRIPTION

vwcvph allows the user to view a convex polyhedron from various positions with hidden surfaces removed. The viewing position is selected using the mouse. Using the SunCore Graphics Package, a 3-D projection is drawn on the Sun Monochrome Bitmap Display. The mandatory file argument contains a polygonal object definition as described in the manual page for **/usr/demo/rotcvph.**

The program operates in two modes: **DisplayObject** mode and **SelectView** mode. The program starts in **DisplayObject mode:**

DisplayObject:
The object is displayed in 3-D perspective with hidden surfaces removed. Type **q** while in this mode to exit the program. Press RIGHT mouse button to switch to **SelectView** mode.

SelectView:
Schematic projections of the outline of the object are shown and the mouse is used to select a viewing position. Press LEFT mouse button to set *x* and MIDDLE mouse button to set *y* in the *Front View*. Use MIDDLE mouse button to set *z* in the *Top View*. Press RIGHT mouse button to switch to **DisplayObject** mode.

The view shown in **DisplayObject** mode is drawn using the conventions that the viewer is always looking from the viewing position toward the center of the object and that the positive *y* axis on the screen is the projection of the positive *y* axis in object coordinates.

The input file may define non-convex objects. The program will display these objects but hidden surface computations will not be done correctly.

FILES

/usr/demo/data/*.dat	sample object definition files

BUGS

Lines which are common to two visible polygons in a view are drawn twice, once for each polygon.

NAME

worm – play the growing worm game

SYNOPSIS

/usr/games/worm [*size*]

DESCRIPTION

In **worm,** you are a little worm, your body is the o 's on the screen and your head is the @ . You move with the hjkl keys (as in the game **snake**). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit; if your worm eats the digit it will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

BUGS

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

NAME

worms – animate worms on a display terminal

SYNOPSIS

/usr/games/worms [**–field**] [**–length** #] [**–number** #] [**–trail**]

DESCRIPTION

–field makes a "field" for the worm(s) to eat; **–trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

FILES

/etc/termcap

SEE ALSO

Snails by Karl Heuer

BUGS

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Terminal initialization is not performed.

NAME

wump – the game of hunt the wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

wump plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it will give a more detailed description if you want.

This program is based on one described in *People's Computer Company, 2*, 2 (November 1973).