NAME

intro – introduction to games

DESCRIPTION

This section describes the recreational and educational programs found in the directory **/usr/games**.

NAME

advent – explore Colossal Cave

SYNOPSIS

/usr/games/advent

DESCRIPTION

Advent is Adventure, the original computer-moderated role-playing game. It accepts commands of one or two English words and responds by describing situations and how your commands affect them. The object of the game is to retrieve the treasures from Colossal Cave, placing them in the Well House.

Part of the game is figuring out the useful commands, but the following are worth knowing in advance:

help Basic hints.

quit End the game and give final score.

suspend Save the game's current state in a file called $HOME/**adv.susp**. The next time you play the game will you automatically start from where you left off instead of from the beginning.

FILES

/usr/games/advfiles/*
$HOME/adv.susp

WARNINGS

Kibitzing this sort of game properly is a fine art. People who tell you about the shortcuts can spoil the game, especially in the early stages.

Some movement verbs, such as **follow**, work only well enough to get you lost. Compass points are more (but not completely) reliable.

Only the first five characters of an input word are significant.

The command vocabulary and control of objects is limited. But discovering limitations has become part of 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 (delete).

The first optional argument determines the kind of problem to be generated; +, −, **x**, and / 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

back – the game of backgammon

SYNOPSIS

/usr/games/back

DESCRIPTION

Back is a program which provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select. In addition to selecting the opponent's level, you may also indicate that you would like to roll your own dice during your turns (for the superstitious players). You will also be given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated.

The points are numbered 1–24, with 1 being white's extreme inner table, 24 being brown's inner table, 0 being the bar for removed white pieces and 25 the bar for brown. For details on how moves are expressed, type **y** when *back* asks "Instructions?" at the beginning of the game. When *back* first asks "Move?", type **?** to see a list of move options other than entering your numerical move.

When the game is finished, *back* will ask you if you want the log. If you respond with **y**, *back* will attempt to append to or create a file **back.log** in the current directory.

FILES

/usr/games/lib/backrules	rules file
/tmp/b*	log temp file
back.log	log file

BUGS

The only level really worth playing is "expert", and it only plays the forward game.
Back will complain loudly if you attempt to make too *many* moves in a turn, but will become very silent if you make too *few*.
Doubling is not implemented.

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".

```
?               (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 (DEL) and the action and standing will be printed.

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), DEL, or control-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

hangman – guess the word

SYNOPSIS

/usr/games/hangman [arg]

DESCRIPTION

Hangman chooses a word at least seven letters long from a dictionary. The user is to guess letters one at a time.

The optional argument *arg* names an alternate dictionary.

FILES

/usr/lib/w2006

BUGS

Hyphenated compounds are run together.

NAME

maze – generate a maze

SYNOPSIS

/usr/games/maze [*seed* [**d**] [**n**] [**b**]]

DESCRIPTION

Maze prints a maze. It uses the system clock as the random number seed. If *seed* is specified, *maze* uses it as the seed and shows the solution. An **n** suppresses the solution, a **b** shows backouts, and a **d** provides debugging information.

BUGS

Some mazes have no solutions.

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

quiz – test your knowledge

SYNOPSIS

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

DESCRIPTION

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

Quiz tells a correct answer whenever you type a bare new-line. 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 new-line | 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 new-lines into a line. When either a question or its answer is empty, *quiz* will refrain from asking it.

FILES

/usr/games/lib/quiz/index
/usr/games/lib/quiz/*

BUGS

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

NAME

trk – trekkie game

SYNOPSIS

/usr/games/trk [[**–a**] file]

DESCRIPTION

Trk 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", "commadore", or "impossible". You should normally start out as 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
cloak up/down
computer request; ...
destruct
help
lrscan
phasers automatic amount
phasers manual amt1 course1 spread1 ...
torpedo course [**yes**] angle/**no**
ram course distance
shell
srscan [**yes**/**no**]
status
undock
warp warp_factor
capture
damages
dock
impulse course distance
move course distance
rest time
shields up/down
terminate yes/no
visual course

NAME

ttt, cubic – tic-tac-toe

SYNOPSIS

/usr/games/ttt
/usr/games/cubic

DESCRIPTION

Ttt is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

Cubic plays three-dimensional tic-tac-toe on a 4×4×4 board. Moves are specified as a sequence of three coordinate numbers in the range 1-4.

FILES

/usr/games/ttt.k learning file

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).

BUGS

It will never replace Adventure.