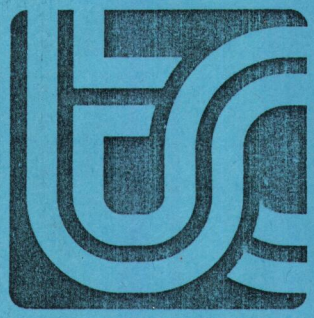
UniFLEX™ Pastime Package 1



technical systems consultants, inc.

UniFLEX™ Pastime Package 1

MANUAL REVISION HISTORY

Revision	Date	Change
A	3/84	Original Release

COPYRIGHT INFORMATION

DISCLAIMER

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc. reserves the right to make changes in such material at any time without notice.

Introduction

The Pastime Package is provided solely for the personal use and enjoyment of the purchaser. No support or maintenance is supplied with the package. The user should not telephone Technical Systems Consultants for technical assistance with the programs in this package. Problems may be reported in writing to the Problem Investigation Department, but no response is guaranteed.

Future releases of the Pastime Package may contain additional games. Because no maintenance is supplied with the package, updated versions will not be supplied for free. Rather, the user will have to return the original master disk and pay a nominal fee to cover the cost of the updating procedure.

Summary of Commands for the Pastime Package

addfortunes	Add new sayings to the data base for the "fortune" command.
battleship	Play a game similar to the board game Battleship™.
blackjack	Play the game of blackjack with the computer as the dealer.
fortune	Randomly display a message from the "fortune" data base.
hangman	Play the word-guessing game of hangman.
life	Play Conway's game of life.
listfortune	Copy the data base for the "fortune" command to the specified file for editing.
makewords	Add words to the data base for the "hangman" command.
mastermind	Play the game of Master Mind™.
othello	Play the game of Othello™.
poker	Play draw poker with the computer acting as dealer.
snake	Race against a snake to get the gold.
spacevoyage	Play Space Voyage™, a space simulation game.

Battleship™ is a trademark of Milton Bradley.
Master Mind™ is a trademark of Invicta Plastics.
Othello™ is a trademark of Gabriel Industries, Inc.
Space Voyage™ is trademark of Technical Systems Consultants, Inc.

Syntax Summaries for the Pastime Package

/usr/games/addfortunes <file_name> [<file_name_list>]

/usr/games/battleship

/usr/games/blackjack [<number_of_decks>]

/usr/games/fortune

/usr/games/hangman

/usr/games/life

/usr/games/listfortune <file_name>

/usr/games/makewords <file_name>

/usr/games/mastermind

/usr/games/othello

/usr/games/poker

/usr/games/snake

/usr/games/spacevoyage

addfortunes

Add new sayings to the data base for the "fortune" command.

SYNTAX

/usr/games/addfortunes <file_name> [<file_name_list>]

DESCRIPTION

The "addfortunes" command adds sayings to the file "/usr/games/data/fortunes", which is the data base used by the "fortune" command. It also reports the total number of sayings in the data base. To add sayings to the data base, the user simply creates a file or files in the format described later (see Arguments) and uses them as arguments to the "addfortunes" command.

Arguments

The files specified as arguments to "addfortunes" must have the following format:

```
The first fortune goes here.  It may have as many lines as
desired.  The saying must appear in the file exactly as
it is to appear on the screen.
@
The next saying goes here.  An "at" sign, '@', in
the first column of a line separates the fortunes.
@
The next saying goes here.  Every saying must be separated
from the preceding one by an "at" sign in the first column
of a line.
The last saying must be followed by an "at" sign, as
shown in this example.
@
```

EXAMPLES

```
/usr/games/addfortunes sayings_1 sayings_2
```

This example adds the sayings in the files "saying_1" and "saying_2" to the data base for the "fortune" command.

(continued)

NOTES

. In order to delete sayings from the data base, the user must use the "listfortune" command.

SEE ALSO

fortune
listfortune

battleship

Play a game similar to the board game Battleship™.

SYNTAX

/usr/games/battleship

DESCRIPTION

The "battleship" command implements a game similar to the popular board game Battleship™. The rating prompt (beginner or master) allows the user to determine how well the computer plays. The user and the computer each command a fleet consisting of five ships: an aircraft carrier, a battleship, a cruiser, a submarine, and a destroyer. The object of the game is to destroy the opponent's fleet first. The "ocean" is an 8-by-8 grid with rows labeled from 'a' through 'h' and columns labeled from 1 through 8. Each player places ships on the grid at the beginning of the game. The locations are, of course, unknown to the opponent. The computer randomly places its ships on a grid, and prompts for the location of each of the user's ships. The user specifies their locations in response to the prompts by indicating the orientation, either vertical ('v') or horizontal ('h'), and the uppermost or leftmost coordinate of each ship. The coordinates are the row letter and column number separated by a comma. Each ship occupies a number of points on the grid as shown:

aircraft carrier	5
battleship	4
cruiser	3
submarine	3
destroyer	2

The number of points is the number of hits the ship can withstand before it sinks.

After the user positions the fleet, the computer displays an ocean grid and asks for approval of the positioning of the fleet. If the user does not approve of the map, the positions of all ships must be respecified.

Next, the computer asks if the user wants the first turn. If not, the computer fires first. When the computer fires, it specifies the destination of its missile with a pair of ocean coordinates (letter, number). The user must tell whether the shot was a hit ('h') or a miss ('m'). If it was a hit, the computer prompts for the type of ship. The user responds by typing the first letter of the name of the ship--for example, 'a' for aircraft carrier, 'c' for cruiser. If the user does not remember the location of the ships, he or she can simply type an 'x' in response to the prompt. The computer then displays a modified map of

(continued)

the user's fleet. Any piece of a ship which has been hit by a missile is replaced by a period, '.'. A plus sign, '+', indicates each spot where the computer unsuccessfully fired.

When firing a missile, the user must specify a pair of ocean coordinates (letter, number). The computer responds with a message stating whether or not the missile hit anything and, if it did, what type of ship it hit. If the user types an 'x' before firing, the computer displays a map of the ocean grid containing its fleet. Appropriate letters indicate sections of ships that have been hit. An asterisk, '*', indicates the destination of each unsuccessful missile.

The computer does not tell the user when one of its own ships sinks; however, it does gloat whenever it sinks a ship.

blackjack

Play the game of blackjack with the computer as the dealer.

SYNTAX

/usr/games/blackjack [<number_of_decks>]

DESCRIPTION

The "blackjack" command plays an interactive game of blackjack with the computer as the dealer and the user as the gambler. The argument <number_of_decks> specifies the number of standard playing-card decks to use in the game. It must be an integer between 1 and 8 inclusive. The default value is 1.

The object of this game is for the player to get a hand with a higher score than the dealer, without getting a score higher than 21. All numbered cards count as their face value, all picture cards count as 10, and aces count as 1 or 11. The game uses Las Vegas rules (see NOTES for exceptions).

For each hand, the dealer (the computer) asks the user for a bet with the prompt "Place your bet (2 - 800):". Bets must be between $2 and $800, in exact-dollar amounts The player receives two cards faceup, and the dealer receives one card faceup and one card facedown.

If the dealer's faceup card is an ace, the dealer asks how much the player wishes to bet for insurance with the prompt "Insurance bet (2 - 200):". By placing an insurance bet, the player is betting that the dealer has blackjack, or a two-card hand valued at 21. The user can decline the insurance bet by typing an 'n' or a carriage return in response to the prompt. Insurance bets must be between $2 and $200, in exact-dollar amounts. If the dealer has blackjack, the dealer pays insurance bets at 2-for-1, the player loses the original bet, and the next hand begins. If the dealer does not have blackjack, the player loses any insurance bet, and the hand continues.

The dealer then requests a command with the prompt "Hit?". The player may request another card (take a hit), double the bet and take a single card (double down), turn a pair into two separate hands (split a pair), or play the hand as delt (stand). The computer expects a 'y' (yes, take a hit), a 'd' (double down), an 's' (split a pair), or an 'n' (no, stand), followed by a carriage return. If the value of the player's hand exceeds 21, the player's hand busts, the player's bet is lost, and the next hand begins; otherwise the computer plays its hand.

The dealer must draw to 16 and stand on 17 (including soft-17, which is 17 with an ace counted as 11). If either the dealer busts or the

(continued)

player's hand exceeds the dealer's hand, the player's bet is paid 1-for-1. If the dealer's hand exceeds the player's hand, the player's bet is lost. If the two hands are equal, the bet is not lost. The dealer begins the next hand.

The player may end the game at any time by typing 'q' in response to a prompt for a bet.

Arguments

<number_of_decks>	An integer between 1 and 8 inclusive indicating the number of decks to use. Default is 1.

EXAMPLES

1. /usr/games/blackjack
2. /usr/games/blackjack 6

The first example plays the game of blackjack using the default number of card decks, 1. The second example plays the game of blackjack using six decks of cards.

NOTES

- Las Vegas rules limit insurance bets to half of the original bet on the hand. This game limits insurance bets to $200, no matter what the original bet was.
- Las Vegas rules only permit one hit on each of a split pair of aces. This game treats a split pair of aces like any other split pair.

fortune

Randomly display a message from the "fortune" data base.

SYNTAX

```
/usr/games/fortune
```

DESCRIPTION

The fortune program displays a saying every time it is invoked. The sayings are not useful, but may be thought-provoking or amusing.

SEE ALSO

listfortune
addfortunes

hangman

Play the word-guessing game of hangman.

SYNTAX

/usr/games/hangman

DESCRIPTION

The "hangman" command implements the familiar word-guessing game. The computer selects a word at random and tells the user how many letters it contains. The user guesses letters one at a time. After each correct guess the computer places the letter in its proper position or positions in the word. If the letter guessed is not in the word, it writes the next letter of the word "HANGMAN" on the screen. To win the game the player must guess all the letters in the word before "HANGMAN" is completely spelled. In other words, on the seventh incorrect guess the computer wins. The player can quit at any time by entering a '#' as the letter guessed.

NOTES

- The "makewords" command is used to add words to the data base.

SEE ALSO

makewords

life

Play Conway's game of life.

SYNTAX

/usr/games/life

DESCRIPTION

The "life" command demonstrates John Conway's cellular automaton ("Scientific American", October, 1970). The user defines the initial population of cells as described in the next section. After describing the population, the user types a control-C to start the game. A control_C is also used to terminate "life".

Defining the Population

The user defines the population by creating a colony of cells. A cell is added by positioning the cursor to the location on the screen where the cell is to be created and typing any character except a control-C, a space, or one of the characters that controls cursor movement. The "life" program displays an asterisk to indicate the presence of a cell. If the character typed is a period, the program does not move the cursor after displaying the asterisk. All other characters cause the cursor to move to the right after the asterisk is displayed. The space character removes a cell.

Five terminal-dependent characters control cursor movement: home up, move left one place, move right one place, move up one place, and move down one place. These characters are are defined in the "termcap" file. A separate document (see Utilities Package II) describes the "crt_termcap" command, which creates the "termcap" file.

NOTES

- The "life" command, which can detect two identical, successive generations, terminates when a static pattern emerges. It cannot detect oscillating patterns.
- The terminal is placed in raw mode while the game is running.

ERROR MESSAGES

Cannot get memory.
The operating system denied a request for memory.

(continued)

Cannot initialize terminal.
The "termcap" entry for the terminal, or the termcap file itself, does not exist.

SEE ALSO

crt_termcap

listfortune

Copy the data base for the "fortune" command to the specified file so that the user can edit it.

SYNTAX

```
/usr/games/listfortune <file_name>
```

DESCRIPTION

The "listfortune" command reads the data base for the "fortune" command ("/usr/games/data/fortunes") and converts it to a text file. The converted output is placed in <file_name>. Once the text file has been created, the user can edit it, adding and deleting fortunes. (Because the data base contains binary information, it cannot be edited directly.) A user who needs to delete fortunes or to add and delete fortunes should first execute "listfortune" to create a text file, then edit the file, delete "/usr/games/data/fortunes", and execute the "addfortunes" command with the name of the new data base as its argument. A user who needs only to add fortunes should use the "addfortunes" command.

Arguments

<file_name>	The name of the file to which to copy the data base, "/usr/games/data/fortunes".

EXAMPLES

```
/usr/games/listfortune ftext
```

This example converts the data base for the "fortunes" command to a text file and puts the text in the file "ftext".

NOTES

- The text form of the file "/usr/games/data/fortunes" has a special format. For a description of this format, see "addfortunes".

SEE ALSO

addfortunes
fortune

makewords

Add words to the data base for the "hangman" command.

SYNTAX

/usr/games/makewords <file_name>

DESCRIPTION

The "makewords" command adds words to the data base for the "hangman" command, "/usr/games/data/hwords". Because this file is not a pure text file, it cannot be edited directly. The first step in adding new words is to use the "edit" command to create a file with one word on each line. The words may not exceed sixteen characters in length. The maximum number of words allowed in the data base is 1024. The program stops adding words when the limit is reached.

Arguments

<file_name> The name of the file containing the words to add to the data base.

EXAMPLES

/usr/games/makewords newwords

This example reads the file "newwords" and adds the words to the file "/usr/games/data/hwords".

ERROR MESSAGES

Words file too large - no words added.
The data base already contains 1024 words, which is the limit.

Can't open words file.
The file "/usr/games/data/hwords" does not exist or cannot be opened.

Word not added - too long.
A word was encountered in the text file which exceeded sixteen characters in length.

SEE ALSO

hangman

mastermind

Play the game of Master Mind™.

SYNTAX

/usr/games/mastermind

DESCRIPTION

The game of "mastermind" is a game of logic. The computer selects a random list of four characters between 'a' and 'f' inclusive. The same character may appear in the list more than once. The object of the game is to determine the exact sequence of characters in the fewest possible moves. The maximum number of turns is fourteen.

Each guess consists of a string of four characters in the range of 'a' to 'f'. After every guess, the computer awards "white stones" and "black stones". The player receives one black stone for every letter guessed which is in the computer's string and in the correct position and one white stone for every letter which is in the string but not in the correct position. For example, if the computer's string is "dabb", and the player guesses "dbaa", the computer awards one black stone (for the 'd') and two white ones (for the 'b' and one of the 'a's). Guessing the correct string results in four black stones and no white stones.

The game continues until the player either guesses the correct string or makes fourteen unsuccessful guesses. The player can end the game at any time by typing a 'g' as the letter guessed.

othello

Play the game of Othello™.

SYNTAX

/usr/games/othello

DESCRIPTION

The "othello" command implements the game of Othello. The user can play the game interactively against the computer, or watch the computer play against itself. In either case, the game can be played at various levels of skill, ranging from novice to very sophisticated.

The board consists of eight rows and eight columns. Rows and columns are both numbered from 1 through 8. The computer displays the initial configuration of the board, which is always the same, and prompts the player for a move.

To understand what a move is, the player must understand the term "outflank". One player outflanks another by positioning a playing piece so that one of his or her pieces is at each end of a row of the opponent's pieces. A "row of pieces" is simply one or more adjacent pieces; it may span the board horizontally, vertically, or diagonally. The same move can outflank more than one row of pieces. When a row is outflanked, each of the outflanked pieces changes to the color of the outflanking pieces. Pieces only change color as a direct result of being outflanked. The player with the larger number of pieces on the board at the end of the game is the winner.

In this game the player places a piece on the board by typing in the coordinates (row number and column number separated by a space) of the square where the piece is to go. A player cannot make a move unless the move outflanks the opponent. If such a move is not possible, the player forfeits the turn.

The game ends when the board is filled, when neither player can move, or when all the pieces on the board are one color. A player may end the game at any time by entering "0 0" in response to the prompt for a move.

The computer issues several prompts before starting to play. The user should respond to these prompts with a 'y' for "yes" or an 'n' for "no". The computer can use the results of a game against itself to modify the algorithm that it uses for selecting its moves. If the user asks it to use previously stored "weights" as it plays, it looks for a file named "othello.weights" in the working directory. If the file exists, it modifies its method of play based on the information in the file. If the file does not exist, the program uses the default algorithm. If the user responds to the appropriate prompt by specifying that the computer

(continued)

should save the "weights" at the end of the game, the computer saves the current algorithm in the file "othello.weights" in the working directory. If the file already exists, the new information replaces the existing data. If the file does not exist, "othello" creates it.

NOTES

- The amount of time the computer uses to make a move can become very great as its skill level increases.

poker

Play draw poker with the computer acting as the dealer.

SYNTAX

/usr/games/poker

DESCRIPTION

The"poker" command simulates the video games of draw poker found in Las Vegas. The player may bet from one to five silver dollars at a time. The rules are those of five-card draw poker, jacks or better to win. The player may draw between zero and five cards.

Each hands begins with a bet. After the bet is placed, the computer deals the player five cards and asks for a list of hold cards (those cards which the player wants to keep). This list must be entered as a string of digits (from 1 to 5) separated by spaces. Each digit represents one of the cards in the player's hand. The string must be terminated with a carriage return. Typing only a carriage return tells the computer to deal five new cards. After the player specifies the hold cards, the computer replaces the other cards with new ones. It automatically determines the value of the hand and pays off accordingly. The following table describes the payoffs.

Table of Payoffs.

Hand	Payoff
Royal straight flush	50 times bet
Straight flush	20 times bet
Four of a kind	15 times bet
Full house	7 times bet
Flush	5 times bet
Straight	5 times bet
Three of a kind	3 times bet
Two pairs	2 times bet
One pair (jacks or better)	1 times bet

The player may end the game at any time by entering a 'q' in response to the prompt for a bet.

snake

Race against the snake to get the gold!

SYNTAX

/usr/games/snake

DESCRIPTION

In this game the user races against a snake towards a "pot of gold" (indicated by a dollar sign, '$', on the screen). The snake, of course, tries to get there first. If the player reaches the gold before the snake, the program creates a new pot, and the race is on again. The player may end the game at any time by reaching the door on the screen, indicated by a pound sign, '#', or by typing a 'q'. The player wins by amassing $250 before being eaten by the snake.

DIRECTIONS

The player is represented by the cursor on the screen, and the snake is represented by a string of 's's with a 'S' as its head.

The player may use the following characters to move the cursor:

h	Move one space to the left.
j	Move one space down.
k	Move one space up.
l	Move one space to the right.
y	Move one space up and one space to the left.
u	Move one space up and one space to the right.
b	Move one space down and one space to the left.
n	Move one space down and one space to the right.
w	Space warp. Moves the cursor to a random position. Costs the player one half of any accumulated gold.

ERROR MESSAGES

Can't initialize terminal - Goodbye!
The terminal environment could not be initialized using the file "/etc/termcap".

SEE ALSO

crt_termcap

Space Voyage™

Play Space Voyage, a space simulation game.

SYNTAX

/usr/games/spacevoyage

DESCRIPTION

Space Voyage is a complex space simulation game. You are the captain in command of the Starship Enterprise and your mission is to rid the galaxy of all enemy Klingon spaceships. The two-dimensional galaxy is divided into 64 "quadrants", eight on a side. Each quadrant is similarly divided into 64 "sectors". A number of Klingon ships are scattered throughout the quadrants of the galaxy, and you must kill them with either your phasers or photon torpedoes before you run out of energy or time. The many resources of the Enterprise are at your command.

The Enterprise's instruments will often inform you of your current quadrant or sector location. Quadrant and sector locations are displayed as two digits in the form, "X-Y". The first digit indicates your horizontal position within the galaxy or quadrant with "1" being the leftmost position. The second digit indicates your vertical position within the galaxy or quadrant with "1" being the uppermost position. Thus, quadrant "1-1" is the upper left-hand corner of the galaxy, and quadrant "8-8" is the lower right-hand corner of the galaxy.

When the game starts, you will be asked if you want to play a short or long game. It is recommended that only short games be played at first since long games tend to be very difficult and can take hours to play. Next, you will be informed of the location of your starbase. It is important that you know its position at all times because you can return to it for refueling. Next, you must enter a three-character password. You will need the password if you wish to execute the self-destruct sequence anytime during your mission.

Finally, a few important lines of information are printed. The first line contains the present stardate, followed by the number of Klingons in the galaxy. You must destroy all of them to save the federation. The next line gives the number of solar years you have to complete your mission. The last two lines inform you of your current quadrant and sector locations in the galaxy.

When your mission actually begins, you will be prompted for commands. Each command is given as a single digit followed by a carriage return. Command numbers may range from 0 through 9. The commands control various devices at your disposal on the Enterprise. These devices, and the commands to control them, are described here.

(continued)

Command 0 - Warp Engines

The warp engines are used to move the Enterprise. If you use this command, you will be prompted for a "course". The course indicates in which direction the Enterprise should move. The Enterprise can only move in eight directions: up, down, right, left, and 45-degree angles between. The course is specified by an integer between 0 and 7. The number 0 specifies a move upward, 1 specifies a 45-degree angle between up and right, 2 specifies right, 3 specifies a 45-degree angle between right and down, and so forth through 7, which specifies a 45-degree angle between left and up. This may be pictorially represented as follows with the Enterprise designated by an 'e'.

```
    0
  7   1
6   e   2
  5   3
    4
```

After specifying the course, you will be prompted for a "warp factor". Your response indicates how far the Enterprise is to move. It can be either an integer (no decimal point) or a fraction beginning with a decimal point (no leading zeros). If the warp factor is an integer, the distance travelled will be that number of quadrants. Fractional warp factors move the Enterprise by one sector for each tenth. For example:

warp factor	movement
.1	1 sector
.5	5 sectors
1	1 quadrant
2	2 quadrants

If you are in quadrant 1-1, sector 1-1 and enter a course of 2 and warp factor of 3, the Enterprise will move to quadrant 4-1. When you change quadrants, the positions of all objects in the quadrant (including the Enterprise) will be randomized to simulate three-dimensional travel. Every use of the warp engines takes time and is proportional to the warp factor used. If the Enterprise is blocked by something during intra-quadrant travel it will stop in front of it and waste time. If the object is a Klingon, the Klingon will be destroyed and the Enterprise badly damaged.

(continued)

Command 1 - Short Range Scanners

The short range scanners display a detailed view of the current quadrant. The Enterprise looks like an 'e' on the screen. Each Klingon appears as a 'k'; the starbase, as a 'b'; and each star, as an asterisk, '*'. To dock with your base, you must first move to one of the eight adjacent sectors, then perform a short range scan. Docking replenishes all energy, shield, and photon levels. While the Enterprise is docked, its shields are lowered, but the shields at the base protect you. The "condition" displayed on this scan can be "docked", "green" (when no Klingons are threatening), "red" (when any Klingons are in the current sector), and "yellow" (when your energy reaches a dangerously low level--300 units or less). If condition yellow exists, you should return to your base immediately.

Command 2 - Long Range Scanners

These scanners display the objects in the eight surrounding quadrants, as well as those in the current quadrant. This information is compiled by the computer and displayed as a 4-digit number for each quadrant. These 4-digit numbers are displayed in a pattern in which the center number represents your current quadrant, and the other numbers are in the proper position relative to that quadrant. The position of each digit within a number indicates a different object as follows:

The	thousand's	digit	is	the	number	of supernovas
"	hundred's	"	"	"	"	" bases
"	ten's	"	"	"	"	" stars
"	one's	"	"	"	"	" Klingons

For example, 0121 means no supernovas, 1 base, 2 stars, and 1 Klingon in the quadrant you are in. If you are at the edge of the galaxy, some of the surrounding quadrants will be displayed as all zeros. A supernova can appear at any time in any quadrant. If it does, all objects in that quadrant will be destroyed. If the Enterprise enters a quadrant containing a supernova, the ship will be destroyed.

Command 3 - Phasers

Any portion of the available energy can be fired using the phasers. The battle computer divides this amount equally among the Klingons in the quadrant and fires at all of them. Each Klingon can be destroyed by an unknown amount of energy (each one may be different). If too little energy is fired, you will only damage the Klingons and will have to fire again. Note: shields must be lowered to fire phasers.

(continued)

Command 4 - Photon Torpedoes

Initially, you will have fifteen photon torpedoes. One torpedo destroys whatever it hits. The range of torpedoes (like phasers) is limited to the current quadrant. The course of a torpedo is set the same way as that of the Enterprise. If the torpedo must travel a long distance to its target, it may run out of energy before reaching it.

Command 5 - Damage Report

The damage report lists the main devices and their states of repair. Devices are only listed if damaged. The number appearing beside the device in the status column indicates the number of solar years which must pass before that device is repaired. Devices are nonfunctioning when damaged. The only exception to this rule is the warp engines, which may still be used, but only for sector moves, when damaged. Damage can occur when a Klingon attacks and your shields are down or when a Klingon is rammed. All damages are repaired when docked.

Command 6 and 7 - Energy Shields

Shield strength is a measure of how much energy the shields can ward off during attacks. It is printed after shield status during a short range scan. When you are attacked, shield strength is weakened. The weakening is cumulative. When the strength reaches 0, the shields are disabled. They can only be repaired by docking with the starbase. Command 6 raises the shields; it uses 200 units of shield energy. Command 7 lowers the shields.

Command 8 - Teleporter

The teleporter instantaneously transports you to your base and operates on energy from the base. It can be used an unknown number of times but not until twelve solar years have passed. After any use it may become damaged, but the damage will not be reported in the damage report. Any time it is used, it may malfunction and place the Enterprise in a random quadrant.

(continued)

Command 9 - Self-Destruct

If you want to give up your mission or decide you cannot save the federation, you may self-destruct the Enterprise. Upon entering this sequence, you will be asked for the password entered earlier. Since you are the captain, you should be the only one with this information. If the wrong word is entered, the sequence will be aborted; otherwise, destruction will result.

When playing, it is a good idea to keep track of all quadrants seen by the long range or short range scan. This lets you keep track of where you have been and where the Klingons are located. A sample grid for this purpose appears on the short-form instruction sheet at the end of these instructions.

Several unexpected events may happen during your mission. A space storm may damage your shields. Supernovas may appear at any time. Some Klingons have the ability to mask themselves from a short range scan (but not a long range scan). You will probably need to use phasers to destroy these Klingons. If you try to leave the galaxy (go beyond the stated boundaries), you will be told 'Galaxy Limit' and halted. If this is attempted three times during your mission, the Enterprise will be destroyed.

--- GOOD LUCK ON YOUR MISSION ---

(continued)

SHORT FORM INSTRUCTIONS

Command list

0 = Warp Engines

1 = Short Range Scan

2 = Long Range Scan

3 = Fire Phasers

4 = Photon Torpedoes

5 = Damage Report

6 = Raise Shields

7 = Lower Shields

8 = Teleporter

9 = Self Destruct

Information kept for the game.

Starbase Location: _____

Password: _____

Initial Stardate: _____

Number of Klingons: _____

Number of Years: _____

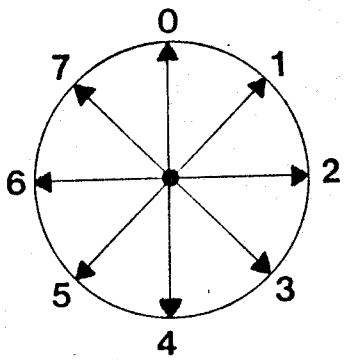
End Time = Stardate + Years: _____

Quadrant & Sector coodinates.

Use during game to keep track of Klingons and Starbase

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

Course direction.



You might wish to make several copies of this sheet.