DOUBLE AUCTION TOURNAMENT

COMPUTER STRATEGY CHALLENGE

REWARDS: $10,000

PARTICIPANT'S MANUAL

Santa Fe Institute

Developed by Richard Palmer,
John Rust and John Miller

PARTICIPANT'S MANUAL

FOR

A DOUBLE AUCTION TOURNAMENT

Richard G. Palmer
John Rust
John Miller

Santa Fe Institute
1120 Canyon Road
Santa Fe, NM 87501

Abstract

This manual describes the Double Auction Tournament being run by the Santa Fe Institute in March 1990. It explains the Double Auction, the tournament, and all the associated software and protocols. Instructions are given for preparing a strategy-playing program to participate in the tournament.

Table of Contents

The documentation is all available electronically, either by anonymous ftp from **sfi.santafe.edu** (directory pub/dat/doc) or on PC floppy disks; the appropriate file names are listed above.

Acknowledgements

We thank all participants in the Economics Program of the Santa Fe Institute for discussions and, in many cases, for being guinea pigs as the Double Auction game developed. We are also very grateful to Robert Axelrod, Robert Dorsey, Stephanie Forrest, Charles Plott, Vernon Smith, and Shyam Sunder for helpful comments and suggestions. We thank Citicorp, the Russell-Sage Foundation, IBM, and the Alfred P. Sloan foundation for financial support. We are also grateful for the superb administrative assistance provided by the Santa Fe Institute, in particular help from Michael Angerman, Ronda Butler-Villa, Shona Holmes, Stephen Pope, Ginger Richardson, Mike Simmons, Andi Sutherland, and Della Ulibarri.

CHAPTER 1: INTRODUCTION

This chapter describes the Double Auction (DA), including the specific conventions of the Santa Fe Institute's implementation. It provides an example of part of a game, raising some of the strategy issues. It also gives a broad overview of how the computerized tournament works, and a guide to the rest of the documentation.

1.1 Background

The Double Auction (DA) is a type of trading institution used by the New York Stock Exchange, the Chicago Board of Options Exchange, and many other securities and commodity exchanges throughout the world. In the DA, buyers and sellers are simultaneously able to call out "bids" (offers to buy) and "offers" (offers to sell), or to accept the lowest outstanding offer or highest outstanding bid at any point in the trading process. The rapid flow of information combined with the ability of traders to undercut instantly any outstanding bid or offer makes the DA perhaps the closest embodiment of the economist's notion of a perfect frictionless market. The efficiency properties of these markets are legendary and have been extensively studied in laboratory experiments using human subjects (see surveys by Plott, 1982 and Smith, 1982a).

In order to gain a deeper understanding of strategic choice in the DA, the Santa Fe Institute is sponsoring a computerized DA tournament. One goal of this tournament is to analyze the game when played by computerized trading strategies. To date, formal game-theoretic analysis of the dynamic DA game has not been forthcoming due to its inherent complexity. Insights gained into the nature of the decision processes actually employed in such situations could ultimately prove useful for improving our understanding of certain market phenomena such as the stock market crash in Fall 1987. Recent interest on Wall Street about the impact of computerized trading programs on market stability makes this analysis particularly timely.

The inspiration for a computerized DA comes in part from Axelrod's (1984) "prisoner's dilemma" tournament. The tournament allows alternative algorithms to compete against one another in a carefully controlled experimental environment. Within this environment, a variety of experiments can be performed on the submitted algorithms. This will allow us to evaluate carefully the important elements of each strategy, and to develop new insights into the game. A priori it is difficult to predict which strategies will perform well. Will complex strategies be more effective than simple ones? Can information about one's opponents be productively exploited? Are there strategies which can perform well in a variety of environments (for example, large versus small markets)? How will the strategies perform against human traders? Can the behavior of the computerized strategies be easily distinguished from human players? How well will the computerized double auction market operate? Will it result in highly efficient allocations as is observed in human DA markets? Will the market quickly stabilize at the competitive equilibrium allocations, or will instabilities such as market bubbles and crashes arise?

To provide strong incentives for the submission of well developed strategies, prize money in the amount of $10,000 will be distributed among the participants. Payments will be proportional to the trading profits earned by the participant's strategy in the tournament. Algorithms are being solicited from individuals with a variety of backgrounds including business school students, computer scientists, economists, game theorists, learning theory specialists, and possibly even traders from real exchanges such as the New York Stock Exchange and the Chicago Board of Trade. It is our hope that these algorithms will embody a variety of strategic approaches, from simple, intuitive "rules-of-thumb" to sophisticated programs utilizing complex statistical, adaptive, and learning algorithms.

The idea of running a Double Auction Tournament originated at the March, 1988 meeting of the Economy Program of the Santa Fe Institute. The original protocol was designed by Richard Palmer and John Rust. The experimental design of the tournament was developed jointly by John Miller, Palmer and Rust, with comments from Robert Axelrod, Charles Plott, Vernon Smith, and Shyam Sunder. A prototype version of the DA software was written in the Gauss programming language (for IBM PC's) by Rust in December, 1988. The software design is due Palmer and Miller. The tournament software was written by Palmer (monitor, DANI, C and Fortran players) and Miller (Pascal players), with assistance and advice from Rust, Stephen Pope, and Michael Angerman. George Tsibouris and Palmer wrote the PC graphics routines using Turbo-C. The documentation was written by Palmer, with assistance from Miller and Rust.

1.2 The Computerized Double Auction

The structure of our computerized DA game is very similar, but not identical, to the version of the DA used in laboratory experiments. The main differences are (1) time is divided into discrete *bid-offer* and *buy-sell* steps, and (2) the adoption of *Chicago rules*. The time discretization was adopted to simplify the synchronization of communications in a multi-processing or network environment with delays that may vary from player to player or moment to moment. Chicago rules specify that whenever there is an outstanding bid or offer only the *current bidder* or *current offeror* can transact (see rules 15 and 16 below). Chicago Rules were chosen to make our implementation of the DA resemble the new AURORA computerized DA market created by the Chicago Board of Trade. Having experimented with alternative sets of trading rules, we found that the Chicago Rules produce a "fairer," more strategically interesting version of DA without much random tie-breaking.

The rest of this section describes the rules for our game:

1. Each player in a game is either a *buyer* or *seller* of tokens. Players can specify which roles they are willing to play. They are all informed as to how many buyers and sellers there are in the game. In the tournament a program that can play both roles will play more games, and hence have more opportunity for profit, than one that can play only one role.

2. A game is divided into one or more *rounds*.

3. Each round is divided into one or more *periods*.

4. In each period, each buyer can buy up to N tokens and each seller can sell up to N tokens. A *token* can be thought of as a fictitious commodity that is being traded in the DA market. For simplicity, N is the same for all players and all periods in a game, and is between 1 and 4, inclusive.

5. For each round, each buyer is assigned N *redemption values*, one for each token that could be bought. If the i'th token bought by a given buyer in a given period has redemption value R(i), and is bought at a price P(i), then that buyer makes profit R(i)—P(i) from the transaction. Thus buyers try to buy tokens as cheaply as possible below their redemption values. Redemption values are given in decreasing order, and are assumed to be used in that order since it maximizes profit.

6. For each round, each seller is assigned N *token costs*, one for each token that could be sold. If the i'th token sold by a given seller in a given period has token cost C(i), and is sold at a price P(i), then that seller makes profit P(i)—C(i) from the transaction. Thus sellers try sell tokens for as much as possible above their token costs. Token costs are given in increasing order, and are assumed to be used in that order since it maximizes profit.

7. Redemption values and token costs in general are different for each player, and are private information not communicated to other players.

8. Note that redemption values and token costs are the same for each period within a round, but in general differ from round to round. During a given round, strategies may be able to benefit from information gathered about opponents' redemption values or token costs in previous periods.

9. Each period consists of a number of alternating *bid-offer* steps and *buy-sell* steps, starting with a bid-offer step.

10. In a bid-offer step each buyer who has not already bought N tokens in that period may make a bid to buy one token. The buyer specifies the price of the bid; prices must be integers. If there is already a current bid (see below) the price must be higher than the current bid price.

11. In a bid-offer step each seller who has not already sold N tokens in that period may make an offer to sell one token. The seller specifies the price of the offer; prices must be integers. If there is already a current offer (see below) the price must be lower than the current offer price.

12. At the conclusion of a bid-offer step, the highest bid made becomes the *current bid*, and the lowest offer made becomes the *current offer*. If no new bids were made the previous current bid remains if there was one; otherwise there is no current bid. Similarly for offers. Any ties for highest bid or lowest offer are broken randomly.

13. All players are informed about all legal bids and offers, and who made them, at the conclusion of each bid-offer step. At all times they know the current bid and offer (if any), and who holds them—the *current bidder* and *current offerer*.

14. In a buy-sell step, one or more buyers may ask to accept the current offer; this is a *buy* request (see also rule 15). Similarly one or more sellers may ask to accept the current bid; this is a *sell* request (see also rule 16). If more than one request occurs, one is selected randomly; only one transaction (either a buy or a sell) can occur in each buy-sell step.

15. If there is a current bid, then only the current bidder is eligible to make a buy request. If there is no current bid, then any buyer who has not already bought N tokens in this period may make a buy request.

16. If there is a current offer, then only the current offerer is eligible to make a sell request. If there is no current offer, then any seller who has not already sold N tokens in this period may make a sell request.

17. When a transaction occurs the transaction price is equal to either the current offer (if a buy request was accepted), or to the current bid (if a sell request was accepted). Both the buyer and seller involved receive profit as described above (see rules 5 and 6), and consume one of their N tokens.

18. All players are informed about all transactions that occur. The information consists of the transaction price, the identity of the buyer and seller involved, and whether it was a buy or a sell request that was accepted. They are not informed about other buy or sell requests that were not accepted.

19. If a transaction occurs in a buy-sell step, then both the current bid and the current offer are cleared before the next bid-offer step. Otherwise they are retained.

1.3 An Example

A simple example will serve to clarify the above definition of the game, and to raise some of the strategy issues.

We focus on a single seller S1 in a game with three buyers (B1, B2, and B3) and three sellers (S1, S2, and S3). We consider only the first few steps of one period of the game. Our seller S1 was assigned four tokens with the following token costs for the round we are examining:

100 200 300 400

The first few steps are shown in the following table, in a format based on the monitor's listfile display (see the "monitor" chapter of the documentation for full details). The table presents the information available to S1. The left-hand columns show the time and step type (BO = bid-offer, BS = buy-sell). The right-hand columns show the current bid and current offer after each step, and the price of each transaction that occurs. The participants in a transaction are shown by >'s (for a sell) or <'s (for a buy).

t	B1	B2	B3	S1	S2	S3	cbid	coff	price
1 BO	50	75*	60	410*	500	450	75	410	
BS							75	410	
2 BO	100	85	130*	350	330*	400	130	330	
BS							130	330	
3 BO	150	140	175*	250*	275	320	175	250	
BS			>	>					175
4 BO	80*	80	60	400	405	250*	80	250	
BS							80	250	
5 BO	125*	100	110	225*	230	240	125	225	
BS							125	225	
6 BO	150		180*	215	200*	215	180	200	
BS			<		<				200

BID-OFFER STEP 1:

The period begins, and our seller is called on to make an offer. She has no information about the other players (other than the **game-type** parameter which tells how tokens have been generated, see section 3.2), and can only judge an appropriate offer value by looking at her own token costs (100, 200, 300, 400). She could do nothing, and see what the others do, but she decides to make an offer of 410. If accepted this would yield her a profit of 410 - 100 = 310.

Her offer turns out to be the lowest among the three sellers (see the table) and so becomes the current offer. On the buyer's side the highest bid was 75, which thus becomes the current bid. The current bid and current offer are marked by asterisks (*) in the table.

BUY-SELL STEP 1:

Because she is the current offerer, our seller is now eligible to make a sell request, to sell her first token to the current bidder (B2) at the current bid price of 75. But that would be foolish; she would make a loss of 25. So she does nothing.

The current bidder also has the option of making a buy request, to buy a token from our seller at her current offer price of 410. She'd be happy if he did so (giving her 310 in profit), but he doesn't.

This illustrates a general feature of the buy-sell step. It can be thought of as a two-person game between the current bidder and the current offerer, and each hopes that the other will accept. Besides locking in a profit, the incentive to make a buy or sell request oneself is mainly not to have to play another bid-offer step, with the consequent risk of losing the bidding or offering lead. There may also be an incentive to make a trade before time runs out towards the end of a period.

BID-OFFER STEP 2

Given that her outstanding current offer of 410 was not accepted, our seller decides to lower her offer to 350. Here she must strike a balance between going too low, and thus losing potential profit, and not going low enough to win the current offer. Ideally she would guess the best offer of the other sellers, and then go just below it (assuming she could still make a profit). As it is, she doesn't go low enough; seller S2 beats her with an offer of 330. The current bid for the buyers increases to 130.

The bid-offer step can be thought of as an n-person game between the n buyers, and an m-person game between the m sellers. Actually it may not always be advantageous to win the bidding. One strategy is to "hang behind" the leaders until a transaction seems imminent, and then to jump in just ahead of the competition. But this requires good knowledge of the others' probable behavior.

BUY-SELL STEP 2

Again no transaction occurs. In fact this is the normal state of affairs; there are typically far more steps in a period than there are tokens to be traded, so transactions occur relatively infrequently. One strategy is to make a buy or sell request as soon as a "reasonable" profit can be made, before "all the best buys are gone." Though this may give early profit in a game, it does not necessarily make the most profit; holding out for a better price may do better.

BID-OFFER STEP 3

Our seller goes down to 250 and again gets the lead. Meanwhile the buyers come up to 175.

BUY-SELL STEP 3

Our seller again has a chance to request a sell, and this time she chooses to do so. It is accepted and she has made a profit of 175—100 = 75.

We don't know whether the current bidder B3 made a simultaneous buy request; if he did there was a random draw, which S1 must have "won." This is a random draw it is not good to win; if B3 had won the draw the transaction price would have been 250 instead of 175, actually doubling our seller's profit.

BID-OFFER STEP 4

Because a transaction occurred, the current bid and offer have been removed (see the right-hand columns). So now S1 can make any bid she likes. One strategy here is to jump close to the previous transaction price; another is to start far away and work slowly back towards it. The latter gives more information, and is perhaps more likely to convince the buyers to accept a higher price, but is thwarted by just one seller employing the "let's get right back near the last price" approach.

Our seller decides to make a high offer of 400. But S3 jumps in with a much lower offer, at 250. Note that on the buyer's side there was a tie for the current bid, and B1 was randomly chosen.

BUY-SELL STEP 4

No transaction occurs.

BID-OFFER STEP 5

Our seller now has a rather limited range in which to make a new offer; she must go below the current offer of 250, but obviously wants to stay above her next token cost of 200 (recall that she sold her first token, and is now onto her second). She splits the difference at 225, and wins the current bid.

BUY-SELL STEP 5

No transaction occurs. Our seller is eligible to accept the current bid of 125, but would make a loss if she did so.

BID-OFFER STEP 6

S1 is pinned between 200 and 225. She tries 215, but is beaten by S2 at 200. Note that now there is nothing she can do until after a transaction occurs; the current offer is too low for her to beat with any hope of profit. Notice that B2 didn't make a bid on this step. This may signify that he cannot do so profitably, but may also represent a wait-and-see approach. It could also be that he didn't respond in time or encountered an error. Watching where other players stop bidding or offering often gives valuable information, but must be interpreted carefully. Bluffing (by sticking at one price, even though it may not be close to the token value) is certainly possible.

BUY-SELL STEP 6

B3 accepts S2's offer and a second transaction occurs. There was a range of only 20 points between the current bid and offer at this point. Patient players might have continued for several more steps, with the prices edging together, before concluding a transaction. As the prices come together, either player can gain more profit by accepting, but both risk losing the bidding/offering lead to other players.

We leave the detailed example at this point. Several more transactions would probably occur during this period. Then the next period would begin, with everyone having the same token values. In this new period our seller might base her offers and sell requests on information gained in the previous period. She might also want to use some of the information she ignored during the first period, including the actual bids and offers (whether accepted or not) made by each of her opponents during the actual play, the transaction prices for the trades that did occur, and so on. She might also be able to develop a sense of how her opponents are likely to respond (such as the observation that S3 always seems to offer 10 points below the previous current offer value when there was one). This sort of information might even be useful in another round, where the opponents are the same even though the token values are different.

One of the best ways to understand the rules of the game, as well as to gain insights into potential strategies, is to actually play some games against a set of sample strategies. This can be done either on the Santa Fe Token Exchange or entirely on a local computer, as described below and in the rest of the documentation.

1.4 Implementation of the Computerized Double Auction

The Santa Fe Institute's implementation of a computerized double auction is based on two principles:

1. The players in a game are represented by separate independent programs. This allows easy development of players in a diverse set of languages on a variety of machines, and makes it easy to construct games between selected subsets of available players.

2. Participants developing strategies only need to understand and work with programming details related to strategy, not with control, communications, book-keeping, synchronization, etc. (This is almost but not quite true.)

Principle (1) is realized by having a separate *Monitor Program* that manages the game. The players and the monitor are all separate programs. The monitor is NOT an auctioneer; it only serves to coordinate message-passing between the separate player programs. Thus, all programs communicate only with the monitor, but not directly with each other. There are various communication methods that can be used, though not all are available on all systems; these are discussed in detail in Chapter 2, Software.

Depending on the system, the monitor and players may all be on the same machine, or may be distributed on a network. At present networking is possible only for TCP/IP communications on the Internet network. Local players (on the same CPU as the monitor) are normally started up and controlled by the monitor, according to a list in the "players file." Network players are started up separately after the monitor is running, and then connect to the monitor.

A networking monitor running regularly at Santa Fe (every hour on the hour) forms the *Santa Fe Token Exchange* (SFTE). Participants with TCP/IP access to the Internet can play practice games with the SFTE across the network. Results are mailed back to them automatically. But even if they can use the SFTE, most participants will want to set up a monitor running on their own computer too. Principle (2) is realized by the provision of "skeleton" player programs in C, Fortran, and Pascal. These programs will run as they are, playing the Double Auction according to a simple strategy. The strategic parts of these programs are well separated from the control and communications parts, so that participants can easily identify the routines they need to modify to implement their own strategy. They should not need to understand the control and communications parts at all.

In all three languages a standard set of variables specifies the state of the game, and standard routines are called to request strategic decisions. For example, in a program playing a buyer, a routine called "bid" is called in each bid-offer step and must return the value of the bid to be made (or 0 for none). Within the "bid" routine the current status of the game, and some cumulated history, is available for reference in making the decision. Chapter 3, The Skeleton Programs, describes the variables and routines in detail.

The initial stages of setting up the Double Auction on a new machine may take a little time, and might in some cases require help from the organizers. But once a monitor and the skeleton programs are running, actual development of a strategy should be simple (at least from a programming standpoint) and can entirely ignore all the communication and control issues.

A special player program called the "human player" provides an interface so that participants can themselves play a game against computer players. This may be done both on the SFTE and with a monitor running locally.

The monitor, skeleton players, and associated support software have been tested and debugged on a variety of machines and systems including:

Unix systems (BSD and System V)
IBM PC's and compatibles
Apple MacIntosh
Vax/VMS
Cray/Unicos

The organizers will advise on porting the software to further systems, and will help with it when appropriate.

All the software is available from the Santa Fe Institute, via electronic file transfer (anonymous ftp or electronic mail), or on floppy disks. Chapter 2, The Software, gives detailed instructions for obtaining it. Knowledgeable ftp users can look directly in pub/dat by anonymous ftp at **sfi.santafe.edu**; the README file contains a description of the files.

The documentation itself is also available in electronic form, or can be obtained in printed form from Santa Fe. A small charge is made for sending disks or printed documentation from Santa Fe. The software itself is free except for that required to set up a networking monitor other than the SFTE; this is not normally distributed (in part to encourage use of a single network token exchange), but may be sold and licensed on request.

The actual tournaments will be conducted at Santa Fe, not over the network. Participants will submit the source of their player programs, or at least the strategic subroutines from those programs. The "rules" chapter of the documentation contains details.

1.5 Electronic Mailing List

An automatic electronic mailing list is maintained at the Santa Fe Institute for discussions and announcements relating to the Double Auction Tournament. You will be added to the list when you make an initial enquiry about the project, or if you ask specifically to be added.

All mail sent to the e-mail address

dat-list@sfi.santafe.edu

will be sent on automatically to everyone on the mailing list. The list is unmediated, so please be polite, concise, and considerate. The DAT organizers are themselves on the list and will respond as appropriate to queries mailed to the list. Individual queries not of general interest should be mailed to **dat**, not to **dat-list**.

For additions, deletions, and changes to the list, please send mail to

dat@sfi.santafe.edu
or dat-list-request@sfi.santafe.edu

but NOT to the list itself.

All correspondence sent to the mailing list is appended to the file "mail" in the pub/dat directory accessible by anonymous ftp at **sfi.santafe.edu**. See Chapter 2 (software) for more information about ftp.

1.6 Documentation

The documentation for the Double Auction Tournament is divided into a number of different chapters. You will not necessarily need all of these. The name shown in parentheses after each heading below is the file name for that chapter in the electronic form of the documentation.

0. Title Page (title)

    0.1 Title
    0.2 Authors
    0.3 Abstract
    0.4 Table of Contents
    0.5 Acknowledgements

1. Introduction (intro) [This chapter]

    1.1 Background
    1.2 The Computerized Double Auction—Game rules
    1.3 An Example, with comments on strategy
    1.4 Outline of software implementation
    1.5 Electronic Mailing List
    1.6 Documentation summary

2. Software (software)

    2.1 Communication methods
    2.2 Outline of available software
    2.3 Obtaining the software and documentation
    2.4 Getting started
    2.5 Getting help

3. Skeleton Programs (skeleton)

    3.1 Introduction
    3.2 Variables in the user routines
    3.3 Working variables
    3.4 Return values

4. The monitor (monitor)

    4.1 Starting the monitor
    4.2 Runtime commands
    4.3 Vax/VMS notes
    4.4 Explanation of listfile output
    4.5 Explanation of logfile output

5. The player programs (players)

    5.1 Starting the player programs
    5.2 Explanation of human player and C-player output
    5.3 Prompts and responses in the human player
    5.4 History display in the human player

6. DANI (dani) [see Note 1]

- 6.1 Communication methods
- 6.2 Starting DANI
- 6.3 Vax/VMS notes
- 6.4 Explanation of display output

7. Tournament rules (rules)

- 7.1 Rules for entries
- 7.2 Registration form

8. Message passing protocol (messages) [see Note 2]

- 8.1 Messages and message packets
- 8.2 Order of packets
- 8.3 Individual Steps of DA game
- 8.4 Special messages
- 8.5 Summary
- 8.6 Code table

9. References (refs)

- 9.1 References

The text of the poster is also available, in file 'poster'. Permission is hereby granted to post this poster, without substantive change, to any bulletin board or news group.

Note 1: The description of DANI in chapter 6 is only needed if you want to use a language other than C and use the SFTE via the Internet.

Note 2: Chapter 8 is a technical document on the messages passed between monitor and players. It is NOT needed for ordinary participants who base their players on the skeleton programs.

CHAPTER 2: THE SOFTWARE

This chapter describes the software available. It also explains how to obtain the software and documentation that you need, and how to start setting it up on your own machine. There are many details in this chapter, but probably only part of it will apply to you.

2.1 Communication Methods

The first key to understanding the software is a knowledge of the communication methods that can be used between the monitor and the players.

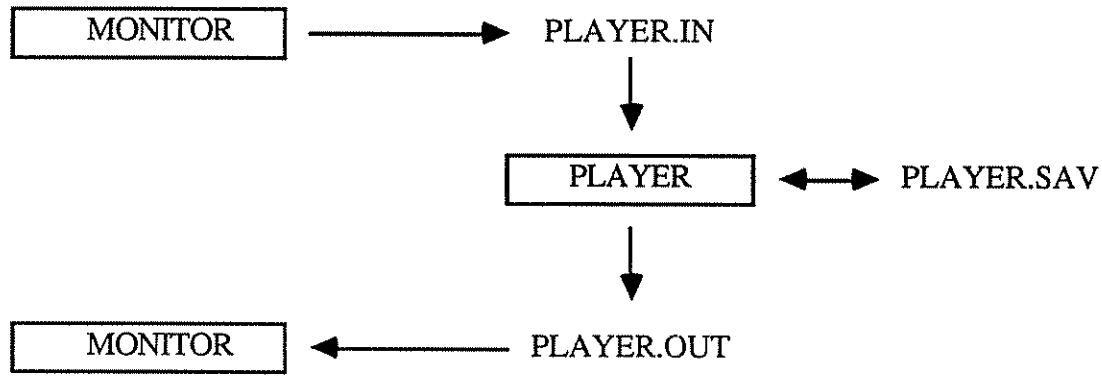
Note that in all games there is one monitor and at least two separate players. The monitor and each of the players is a complete program in itself; the players are NOT "subroutines" of the monitor. The monitor program and the player programs may run simultaneously (either on the same computer or on different ones connected by the Internet), or one at a time. The players each communicate only with the monitor, not directly with each other.

The different communication methods are described below. In each case we show only one player communicating with the monitor, but in a real game there would be more. The monitor can support any mixture of communication methods that are available on the computer concerned.

Certain facilities are only available to player programs based on the skeleton programs written in C. Such programs are referred to as "C players" below, and include the human interface.

FILE-BASED COMMUNICATION

This is the slowest communication scheme but is available on all supported systems. It is the ONLY communication method available on an IBM-PC, or on any other machine that can only run one program at a time. Suppose the player program is called PLAYER. Then the following sequence occurs for each step of the game:



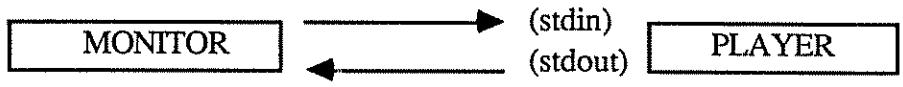
The monitor writes file PLAYER.IN and then starts up the PLAYER, which reads it. The player writes file PLAYER.OUT, which is then read by the monitor. The monitor itself is suspended while the PLAYER is running. The PLAYER is started up separately for each step of the game, and exits after writing PLAYER.OUT. Because the PLAYER is not running continuously it must

save its working variables, either in a file called PLAYER.SAV as shown, or (C players only) in PLAYER.OUT following the output messages.

This communication method is obviously very disk intensive, and also involves many program launches. It is not recommended when other methods are available. On an IBM-PC it can actually be made fairly fast by placing the player programs (and thus their .IN, .OUT, and .SAV files) on a RAM disk. See the instructions below. On a Vax/VMS system it is extremely slow.

PIPE-BASED COMMUNICATION

This is the recommended communication method for use between the monitor and each player when on the same multiprocessing machine. The monitor and players all run simultaneously, and communicate via "pipes" (or their equivalent, e.g., "mailboxes" in VMS):



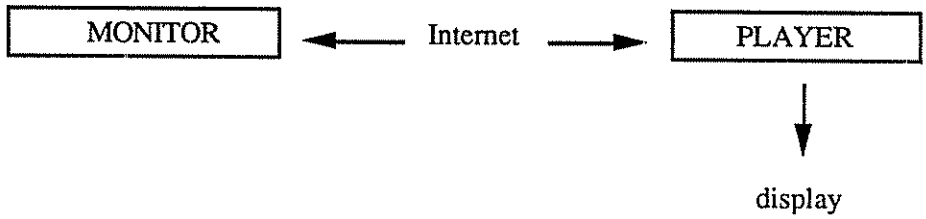
There is one pipe for each direction. At the player's end these pipes are arranged to look like the standard input and output channels, which would normally be attached to the terminal. Thus the player program can use simple read and write operations without any special knowledge of pipes.

The monitor program is started up first and then starts up all the pipe-based player programs. They are technically "child processes" (Unix) or "sub-processes" (VMS) of the monitor's own process. The players keep running until the end of the game or an error; for most errors detected by the monitor the offending player will be "killed."

There is a modified form of pipe-based communication available only for the C players and the human player. The player uses special channels ("descriptors") for the pipes, leaving its standard input and output free for terminal communication. This is essential for the human player. This form of communication is called "arg-based communication" because the descriptor numbers are passed via command-line arguments from the monitor to the player when the player is started.

INTERNET-BASED COMMUNICATION—DIRECT

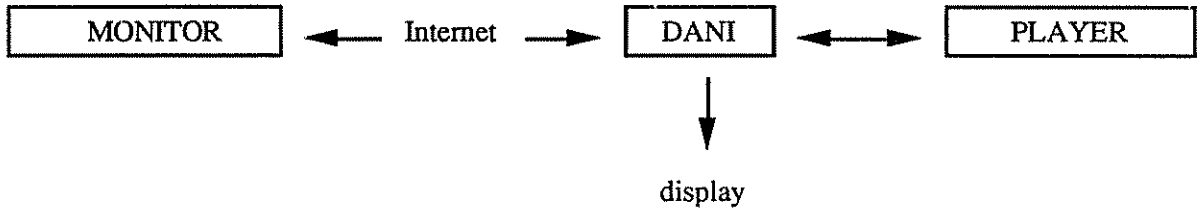
Direct Internet communication is available only for the C players and the human player. It allows such players to connect to the monitor over a TCP/IP Internet communication link:



The monitor and player programs are usually on different machines at different locations. The monitor is normally the one at **sfi.santafe.edu**, called the Santa Fe Token Exchange (SFTE). The monitor runs at a prearranged time (every hour on the hour for the SFTE) and users start up their players at the appropriate time at the remote sites. The player program itself establishes the connection to the monitor. The player program can display the progress of the game to the screen (or to a file) during the game.

INTERNET-BASED COMMUNICATION—VIA DANI

The direct Internet-based method described above is only available for the C players and human player. For other languages the Double Auction Network Interface (DANI) must be used:



DANI acts as a relay for messages between the player and the Internet. It can also display the progress of the game as it runs. DANI and the player can communicate by any of the non-Internet methods described above (file-based, pipe-based, or arg-based). The monitor is started first (at a prearranged time), and then DANI is started at the remote site. DANI then starts the local player. DANI can only support one player at a time.

DANI can also communicate with the player by some additional methods not discussed here. Please contact the organizers if you have a special need not covered by the cases presented here. Using DANI may sometimes be preferred over the direct Internet method even for C players, because DANI's display format is more compact than the C player's (which is designed primarily for the human player).

2.2 Outline of the Software Available

The software available for the Double Auction Tournament consists of the following major parts:

1. The monitor. This is needed if you want to run double auction games on your local machine or network. Without it you can only play games on the Santa Fe Token Exchange (SFTE), via the Internet. This is available both in C source form and in IBM-PC executable form.

2. The skeleton players. You will certainly need one of these, in source form, if you plan to develop your own strategy (without it you can only play via the human interface on the SFTE, and cannot enter the tournament). They are available in the following languages:

C	for Unix, Vax/VMS, Unicos, Turbo-C, Quick-C
Fortran	for Unix, Vax/VMS, Unicos, Microsoft, Ryan-McFarland, Lahey
Pascal	for Unix, Vax/VMS, Turbo-Pascal.

   Using another language (e.g., Lisp, Ada, Basic) is possible for games on the SFTE (via DANI), but is not acceptable for tournament entries except by prior arrangement with the organizers.

   Using one of the above languages on a different system or compiler will not be difficult in most cases, though some features may be lacking. Contact the organizers for advice.

3. The human interface. This allows you to play against computer strategies, either with your local monitor or on the SFTE. The human player is available in C source form and in IBM-

PC executable form. The C source actually shares much of the code of the C skeleton player and is packaged with it.

4. DANI: the Double Auction Network Interface. This is needed if you want to play on the SFTE using a player program based on the Fortran or Pascal skeleton players (or anything else but C). It may also be useful even with the C players because its output display is compact. DANI is available in C source form only.

5. PP: the simple pre-processor. This is a program that processes another program performing file inclusion, parameter substitution, and conditional compilation (it recognizes a subset of the C preprocessor directives). It is needed if you want to use the Fortran skeleton player on a non-Unix machine. It may also be useful for Pascal; see the README file for Pascal. PP is available in C source form and in IBM-PC executable form.

6. Networking software. The programs described above have only partial networking ability. The C players and DANI can contact the Santa Fe Token Exchange only, and the monitor supports only local players (on the same CPU). Only one human player at a time can play in a local game, because it needs the login terminal. To relax these restrictions, either to set up a networking monitor (your own token exchange), or to connect to a token exchange other than the SFTE, you need the full networking software. THIS IS NOT NORMALLY DISTRIBUTED FREE TO PARTICIPANTS. Contact the organizers for further information.

7. Getgame routines. There are C subroutines designed to read the monitor's output files for use in constructing programs for analysis or display of game results.

Note that while the skeleton players are available in several languages, the rest of the software is only available in C or in IBM-PC executable form. However you should not need to understand or modify any of this C-only software—you will just need to compile it once on your system. If you don't have a C compiler please contact the organizers; we will attempt to supply you with a binary executable for your machine.

2.3 Obtaining the Software and Documentation

There are three basic ways to obtain the double auction software:

1. Copy it electronically from the public directories of the **sfi.santafe.edu** computer at Santa Fe, using 'ftp'.

2. Request floppy disks from Santa Fe. This is most appropriate if you intend to run the software on an IBM-PC or compatible.

3. Request that the software (and optionally the documentation) be sent to you by electronic mail from Santa Fe.

The documentation is all available in electronic form (as simple ascii files), and can be included with any of the above methods. Alternatively you can request printed documentation from Santa Fe.

These methods are described in more detail in the following subsections.

OBTAINING THE SOFTWARE AND DOCUMENTATION WITH FTP

To obtain files from Santa Fe via ftp, follow the following steps:

1. Connect to **sfi.santafe.edu** with ftp. On most computers the command "ftp **sfi.santafe.edu**" will do this.

2. Specify the user name "anonymous."

3. Specify your own last name (up to 8 characters) as the password. You should then get ftp's prompt (usually "ftp> ").

4. cd pub.

5. cd dat.

6. If you plan to get the .tar.Z files or dos files (see below): binary.

7. Use a combination of the following commands to get the files you need:

   a. cd <directory> to change to <directory> on the remote machine.

   b. get <filename> to copy <filename> to your machine.

   c. mget <wildname> to copy all files matching <wildname> to your machine. E.g., *.c matches all .c files.

   d. prompt to turn off prompting for each file; use before mget if you want all matches.

   e. ls to list the files in the current directory.

   f. lcd <directory> to change your local directory.

8. quit.

The organization of the directories and files in the pub/dat directory is as follows (names with lines descending beneath them are directories):

```
pub/dat
    |
    |-- README   (contains this chart and Notes)
    |-- README.MAC (special notes for Macintosh version)
    |-- directory (list of participants)
    |-- directory (list of tournament inquiries)
    |-- mail (correspondence sent to dat-list mailing list)
    |-- monitor.tar.Z
    |-- monitor
    |     |
    |     |-- *.c and *.h source files for monitor
    |     |-- Makefile, dmon.mak, dmon.prj
    |     |-- game, players
    |     |-- README
    |
    |-- players
    |     |
    |     |------ C.tar.Z
    |     |------ C
    |     |     |
    |     |     |-- *.c and *.h files for C players and human player
    |     |     |-- Makefile, *.com, *.mak, *.prj
    |     |     |-- helpfile (for human player)
    |     |     |-- README
    |     |
    |     |---- fortran.tar.Z
    |     |---- fortran
    |     |     |
    |     |     |-- *.F and *.h files for Fortran players (needs pp)
    |     |     |-- Makefile, Makefile.cray, make.com
    |     |     |-- README
    |     |
    |     |---- Pascal.tar.Z
    |     |---- Pascal
    |           |
    |           |-- skeleton.pas for Turbo Pascal (IBM PC)
    |           |-- skeleton.p for Unix Pascal (pc)
    |           |-- skeleton.vms for Vax/VMS Pascal
                |-- skeleton.mac for Lightspeed Pascal (MacIntosh)
    |           |-- skeleton.generic for any Pascal (needs pp)
    |           |-- Makefile
    |           |-- README
    |
    |---  misc
    |     |
    |     |---- dani.tar.Z
    |     |---- dani
    |     |     |
    |     |     |-- dani.c, inet.c (source files for dani)
    |     |     |-- Makefile, make.com
    |     |     |-- README
    |     |
    |     |---- getgame.tar.Z
    |     |---- getgame
    |     |
    |           |-- getgame.h, getgame.c
    |           |-- getgame.doc (documentation for getgame)
    |           |-- plotgame.c, plotgame.prj (for Turbo-C only)
```

Continued

```
|       |
|       |---- pp.tar.Z
|       |---- pp
|               |
|               |--- *.c, *.h source files for pp
|               |--- Makefile, make.com
|               |--- README
|
|---    doc.tar.Z
|---    doc
        |
        |---- individual chapters of the documentation
        |
        |---- dos
        |---- dmon.exe
        |---- gdmon.exe
        |---- human.exe
        |---- player exe
        |---- pp.exe
        |---- plotgame.exe
        |---- *.bgi and *.chr files for gdmon and plotgame (PC graphics)
        |---- player (dos version of 'player' gile)
|---    mac.sit (Macintosh files for Stuffit archive--Use BINARY mode)
```

Notes on the above:

1. The .tar.Z files are compressed files that contain the whole directory listed immediately below them. These are only useful if you have a Unix system with 'uncompress' and 'tar'. If you have these you can do the following:

   a. Get the .tar.Z files you need USING THE BINARY MODE of ftp.

   b. Uncompress each file (e.g., "uncompress monitor.tar.Z").

   c. Unpack each file with tar (e.g., "tar xvpf monitor.tar"). This will make the directory concerned (e.g., monitor) within your current directory, and fill it with the files.

   d. Delete the .tar file.

2. Most directories contain a README file. Read this before attempting to compile.

3. Many directories contain files with scripts for compiling and linking. 'Makefile' is for Unix. A .prj file is for Turbo-C. A .mak file is for Quick-C. A .com file is for Vax/VMS. Read these files before using them; there may be things to customize.

4. The dos directory contains executable binaries and graphics files for an IBM-PC. See the following section for more details on these files. Be sure to use the binary mode of ftp to copy such files. They can then be transferred to a PC in several ways (e.g., PC-NFS). If your interest is just in the PC version, you'll probably find it more convenient to get floppies than to try this.

5. 'mac.sit' contains executable binaries and project files for a Macintosh. Be sure to use the binary mode of ftp. You need Stuffit to unpack the file. Alternatively you can request a Macintosh disk from SFI—use the same addrss and fee as for IBM floppies below.

6. The getgame directory contains routines in C for reading and decoding the monitor's listfile output. These may be used to build an analysis or display program. See the getgame.doc file for details. Also included is the source for PLOTGAME, and IBM-PC program that can read games and display them graphically (executables is in dos directory).

OBTAINING THE SOFTWARE AND DOCUMENTATION ON FLOPPY DISKS

If you use an IBM-PC or compatible, it is best to obtain the software on floppy disks. You may alternatively be able to use the ftp method, but will then have to compile everything rather than receiving executables.

Ask for floppy disks by writing to:

Double Auction Tournament
Santa Fe Institute
1120 Canyon Road
Santa Fe, NM 87501

Include media/shipping fee of $3 for floppy disks only, or $10 for floppy disks plus complete printed documentation ($7.50 for documentation only). The full documentation is included on the floppy disks (as simple ascii text files) and is also available via ftp. Unless you make a special request, you will receive two floppy disks containing the following:

Disk 1:

DMON.EXE	monitor without graphics
GDMON.EXE	monitor with graphics
*.BGI, *.CHR	graphics files used by GDMON.EXE (not needed if you have Turbo-C on your hard disk)
HUMAN.EXE	interface to a human player
HELPFILE	file of help information used by HUMAN.EXE
PLAYER.EXE	sample player program (based on the C skeleton)
GAME	sample gamefile, specifying game parameters
PLAYERS	sample playerfile, specifying players in a game
PP.EXE	simple preprocessor
README.PP	instructions for PP.EXE

PASCAL.DIR directory containing the source for the pascal skeleton player:

README	information about the Pascal versions
SKELETON.PAS	Pascal source for the PC

Disk 2:

C.DIR directory containing the source for the C skeleton players, and the human player:

README	instructions for compilation etc.
SKELETON.C	skeleton player
HUMAN.C	human player
CONTROL.C	control routines for skeleton & human players
DEFINE.H	header file for skeleton & human players
*.MAK	Quick-C makefiles
*.PRJ	Turbo-C makefiles
TCCONFIG.TC	Turbo-C configuration file

FORTRAN.DIR directory containing the source for the Fortran skeleton player:

README	instructions for compilation etc.
SKELETON.F	skeleton player
CONTROL.F	control routines for skeleton
GLOBAL.H	header file
SIXCHAR.H	file to reduce variables to six characters, if necessary
F77L.FIG	configuration file for Lahey Fortran

DOC.DIR directory containing all the documentation (except that for DANI) in simple ascii text files.

Note that this does not include:

1. The source files for the monitor.
2. The source files for PP.
3. The source files and documentation for DANI.
4. Certain files not needed on a PC, but which may be needed or useful if you are merely using PC-floppies to transfer the software to another machine.

You should not need any of these for use solely on a PC. If you request it specially, we will send you "Disk 3" containing all the above files, including a README file that explains its contents. Please add $1 for this.

The documentation and source code for "GETGAME," the C program which reads and analyzes the monitor output is available on "Disk 4." Also included is a compiled, binary executable program "PLOTGAME.EXE" which reads the monitor output and produces a simple visual summary graph of the DA game. This program will work on IBM-PC's and compatibles with graphics capability. It is particularly useful for interpreting the monitor output of SFTE games (which is automaticlaly mailed back to participants at the end of the game). Please add $1 for disk 4.

All floppy disks will be 5.25 inch, double-sided, double-density, 362K.

OBTAINING THE SOFTWARE AND DOCUMENTATION BY ELECTRONIC MAIL

This method should only be used if neither of the other two is available. Send your request to:

dat@sfi.santafe.edu

specifying:

1. Your name and institution.

2. Your e-mail address.

3. The type of machine(s) on which you want to run the Double Auction Tournament software. Specify the operating system (e.g., Unix, VMS) too.

4. The language(s) you want to use for your players (C, Fortran, Pascal).

5. Whether you want the source for the monitor. If you don't get this you can only play with the Santa Fe Token Exchange.

6. Whether you want to be able to play games with the Santa Fe Token Exchange. This is only possible if your site has the appropriate TCP/IP software and hardware capability.

We will then send you the files we think you need. No individual file will exceed 50K. Files will be bundled together into .shar files on a Unix system; instructions will be included. Note that binary files cannot be sent via ordinary e-mail.

2.4 Getting Started

There is a lot of software available, and a lot of documentation. You will not master it all at once. We suggest the following approach:

FOR UNIX, VMS, ETC. USERS (MULTIPROCESSING MACHINES)—SFTE USAGE

Before you do anything, try to determine if your machine has Internet TCP/IP access using BSD 4.2 or 4.3 calls. If it doesn't you cannot use the Santa Fe Token Exchange; go to the next subsection. If you can use "telnet" and "ftp" on your machine you probably have the necessary access. To find out for sure, determine whether the following routines are available in your C run-time library:

socket()	connect()	inet_ntoa()	
htonl()	htons()	bcopy()	bzero()

The ones on the first line are the truly essential ones. They may be in a special library. First try to find them via 'help' ('man' on a Unix system). Then, if that fails, ask an expert. Note that for Vax/VMS there are several implementations of TCP/IP; the Double Auction software currently only supports the Wollongong WIN/TCP one. If you have a different TCP/IP implementation and want to use the SFTE, please contact the organizers.

If you do have the appropriate TCP/IP access, you can follow the following suggestions to play a trial game with the SFTE. Otherwise, or if the TCP/IP access is dubious or needs work, proceed to the next subsection instead.

1. If you don't have printed documentation, print the appropriate files from the 'doc' directory. To start with you'll want at least "intro," "software," and "players."

2. Get the source files for the C players (even if you ultimately want to use Fortran or Pascal).

3. Read the README file for the C players and follow the directions therein to compile and link the players using INETBASED; compile at least "inquire" and the human player (called "hi" or "human" depending on your system). If you have difficulty just try to compile "inquire." If that doesn't succeed please check that you followed the instructions (e.g., did you customize define.h?) and then contact the organizers (see "Getting Help" below).

4. Try running "inquire." If it succeeds in connecting to the SFTE, you should get a message telling you about the schedule of games at Santa Fe, or a message that says a game is about to begin. If instead it says (possibly after up to a minute or so of trying) that it couldn't connect to **sfi.santafe.edu**, the problem may either be with the network or with your machine's access to it. Try again later, and keep on trying for a day or more before concluding that there's a real problem. If you have the "ping" command, try "ping **sfi.santafe.edu**"—if this responds with "**sfi.santafe.edu** is alive" then inquire should work too. Asking a network expert may also help.

5. If "inquire" works, try running the human player. Unless you use it at the right time (just before any hour), it should give the same message as inquire. Using it just before the hour should allow you to play a game with the SFTE. Study the "intro" and (at least briefly) "players" chapters of the documentation before attempting to play, but don't be afraid to experiment. Use the "quit" command when you tire of playing. As long as you completed at least the first period, you should receive a listing of the whole game by electronic mail after it is over. Study this in conjunction with the "monitor" chapter of the documentation. Please

tell the organizers if you don't receive a game listing by electronic mail within a few hours of playing a game.

6. You can also try compiling and running a skeleton player (called "di" or "skeleton" depending on your system). This is actually simpler than the human player. When you run it at the right time it should play a game with the SFTE, displaying it as it goes.

7. You now have three directions in which to proceed:

   a. If you want to write in C, and only want to use the SFTE (not a local monitor), study the skeleton.c source file in conjunction with the "skeleton" chapter of the documentation.

   b. If you want to write in Fortran or Pascal, and want to use the SFTE, now is the time to compile and test DANI and the skeleton player in your chosen language. See the appropriate README files and the "dani" chapter of the documentation.

   c. If you want to set up a local double auction monitor on your machine, follow the suggestions in the next subsection.

FOR UNIX, VMS, ETC. USERS (MULTIPROCESSING MACHINES)—LOCAL MONITOR

In some ways this is the most complicated case because you must compile both the monitor and at least one player before you can get anything going.

1. If you don't have printed documentation, print the appropriate files from the 'doc' directory. To start with you'll want at least "intro," "software," "monitor," and "players."

2. Get the source files for the monitor and for the players in the language of your choice. We recommend that you put the player files in a subdirectory of the directory containing the monitor files.

   If your language is not C, also get the C source for the human player. If your language is Fortran, also get the source files for 'pp' if you don't have a standard Unix system (see the Fortran README file). Make sure you also have any associated README files and makefiles. Also make sure you have 'game' and 'players' for the monitor and 'helpfile' for the human player.

3. Compile the monitor. See the README file for directions. Try running it without any command-line arguments: it should read and list the parameters in the 'game' file, and then stop with a 'no active buyers or sellers' message. If anything else happens, investigate before proceeding; first check that 'game' and 'players' are in the monitor's own directory and contain sensible parameters ('players' should not yet define any players).

4. Compile the skeleton player in the language of your choice. See the appropriate README file for directions. If you choose Fortran you may first need to compile 'pp'. You will have to decide on your communication method, either FILEBASED or PIPEBASED; we recommend PIPEBASED for all Unix and VMS systems. For other systems if you are the first person to try this software, it may be easier to start with FILEBASED.

5. Make a second copy of the compiled skeleton program with a different name (e.g., player2 or player2.exe as appropriate). Simply copying it will work unless you are using FILEBASED communication and the program name has to be built into the program, in

which case you'll have to recompile with the new name; see the appropriate README and source files for details.

6. Edit the 'players' file (in the monitor's directory) to specify your two players. See the comments in the 'players' file and in the "monitor" chapter of the documentation.

7. Run the monitor. Now a game between your two players should commence. Note though that games with only one buyer or one seller are not representative of the full game; many strategy issues do not arise.

8. Play with the following to familiarize yourself with the system:

   a. Adding more players. Note that on many systems you can get away with using the same executable binary player program more than once in a game (simply specify it multiple times in the 'players' file). This will never work for FILEBASED players though; these must always have unique names because the *.in and *.out message files must be unique.

   b. Changing the game parameters in the 'game' file.

   c. Controlling the monitor with single letter runtime commands (see the "monitor" chapter of the documentation.

   d. Sending the monitor listfile output to a file (this gives a different screen display); see the "monitor" chapter of the documentation.

9. Compile the human player. See the directions in the README file from the C source. Note that you need to customize 'define.h' differently for the human player and C skeleton players. Define the human player in the 'players' file, being sure to include the -h flag, as well as one or more skeleton players. Run the monitor, sending the listfile output to a file. You should then find yourself playing a game against the skeleton player(s).

10. To start developing your own strategy, study the "skeleton" chapter of the documentation, and then look at the appropriate skeleton.* source file and the README file in the source directory.

FOR PC USERS WITH THE SOFTWARE ON FLOPPY DISKS

1. If you don't have printed documentation, print the appropriate files from the DOC directory on Disk-2. To start with you'll want at least "intro" and "software."

2. Copy the root level files on Disk-1 (i.e. all but PASCAL.DIR) to a suitable directory on your hard disk. Make a subdirectory of that directory for the skeleton programs in the language of your choice (C, FORTRAN, or PASCAL), and copy the contents of the appropriate directory on Disk-1 or Disk-2 into it.

3. As a trial, you can play a simple game, pitting yourself (via the human player interface) against PLAYER.EXE. Get into the directory with DMON.EXE in it, and type:

DMON OUTFILE

A game should begin with you randomly either a buyer or seller. It will be VERY SLOW, and NOT REPRESENTATIVE of the full game with many players; games with only a single

buyer or a single seller (or both) do not raise most of the strategy issues of a full game. But it may help to introduce you to the game and the software. Here are some things to try:

a. Type a ? at any prompt.

b. Type 'help' at any prompt. Try some of the commands/options listed there, especially u, w, and h.

c. Type 'quit' when you are tired of playing.

d. After playing, display or print the file OUTFILE, to which the monitor sent its output.

For more information about the human player and its commands and display, look in the 'players' chapter of the documentation. For more information about the monitor and its output (as in OUTFILE) look in the 'monitor' chapter of the documentation.

4. To make things reasonably fast on a PC, we highly recommend creating a RAM disk for the players. If you do not currently have a RAM disk running on your machine, one can easily be installed using the VDISK program supplied with later versions of DOS. In your CONFIG.SYS file (found in the root directory) insert the following line:

DEVICE=C:\DOS\VDISK.SYS 128

This assumes that your vdisk.sys file is in the C:\DOS directory; modify if necessary. The 128 gives you a 128K RAM disk, which should be sufficient for several players and their message files; increase or decrease as needed. If you have extended memory (beyond 640K) you can install your RAM disk there by adding /E to the end of the above line.

Reboot your machine after changing CONFIG.SYS; you should now have a RAM disk as drive D. You can use it just like any other drive except that its contents are lost when you turn your machine off or reboot.

Copy your players (.EXE files) to your RAM disk, and change your PLAYERS file to reflect their new location; the directory column should just contain 'D:' (or other appropriate drive letter) unless you make subdirectories on your RAM disk.

You can also copy the monitor and its associated files to the RAM disk if you make it large enough, but this will not convey much advantage.

Once you have mastered using a RAM disk for your players, the process of copying the appropriate files to it can be automated with batch file.

5. To try a game with more players, make copies of PLAYER.EXE with different names (e.g., PLAYER1.EXE etc.), and modify the PLAYERS file appropriately to define the players in each game. You can try games with or without a human player. If you include a human player you must direct the monitor's output to a file (e.g., 'DMON OUTFILE'); without a human player you can let the output go to the screen (e.g., 'DMON'). If you don't have a human player the monitor will respond to several single-letter commands from the terminal; see the "monitor" chapter of the documentation.

6. To try a game with graphical display of the supply and demand curves and transaction prices, use GDMON instead of DMON. See also the discussion of the -g and -p options in the "monitor" chapter of the documentation.

7. You can also try changing the game parameters in the file GAME.

8. To start developing your own strategy, study the "skeleton" chapter of the documentation, and then look at the appropriate SKELETON.* source file and the README file in the source directory. You should try compiling the skeleton without any changes (besides the customization required—see the README) before developing any new strategies. The resulting player program should behave essentially like the PLAYER.EXE provided.

9. There is a considerable advantage to having a small executable player file, since it must be loaded frequently. You may want to explore ways provided by your compiler to minimize the .EXE size, including inhibiting some forms of debugging. Some hints are provided in the README files and makefiles. In general we find that Fortran executables are larger than C executables, which are larger than Pascal executables.

2.5 Getting Help

We will try to assist you in getting our software to run on your system if necessary. We request, however, that you make an effort to thoroughly understand this documentation before requesting individual help from us, since many standard questions are already answered in the documentation. Many standard questions are already answered by the organizers in the "mail" file available by FTP from pub/dat (see section 1.5 and section 2.3 for details). The organizers are available to answer questions on the operations of the monitor (pipe-based and file-based only), the skeleton players in C, Fortran, or Pascal, DANI, and PP. You can contact us by mail, phone, fax, or e-mail. E-mail is recommended, particularly because the expert you need may not be physically at Santa Fe, but will receive your message elsewhere. In all cases, please make it clear that you are referring to the Double Auction Tournament. Please realize that the organizers have full-time positions elsewhere, and so one shouldn't expect our technical help to have as rapid a turn-around time as for commercial software.

Address: Double Auction Tournament
Santa Fe Institute
1120 Canyon Road
Santa Fe, NM 87501

Phone: (505) 984-8800 (office hours 8:30 am-5:00 pm MST)

Fax: (505) 982-0565 (24 hours)

E-mail:

dat@sfi.santafe.edu	(for all general inquiries. Do NOT send initial inquiries to "dat-list" since this is a mailing list that goes out to hundreds of people. Send mail to "dat-list" only after reading section 1.5)
palmer@sfi.santafe.edu	(for C, Fortran, monitor, DANI, PP)
miller@sfi.santafe.edu	(for Pascal)
jpr@sfi.santafe.edu	(for questions on DA rules, strategy, etc)

CHAPTER 3: THE SKELETON PROGRAMS

This chapter describes the structure of the skeleton programs and the variables defined therein. The description here is largely language- independent; language-dependent details may be found in README files and/or as comments in the software provided for each available language.

3.1 Introduction

Skeleton player programs are presently available in C, Fortran, and Pascal. The skeleton programs take care of all the bookkeeping and communication details, allowing participants to concentrate on writing the central decision-making routines to implement their strategy. The skeleton programs provided are actually complete programs with a simple built-in strategy, and may be compiled and used as test players. To produce their own player programs, participants will modify the skeletons and re-compile.

A complete player program based on one of the skeleton programs consists of the following parts:

1. The strategy routines, 'BID' and 'BUY' for a buyer, or 'OFFER' and 'SELL' for a seller. These routines return a value indicating the user's decision. In the skeleton programs as supplied these routines contain simple algorithms as examples.

2. Miscellaneous user routines that are called at different stages of the game to provide information to the user. These are all empty dummy routines in the skeleton programs as supplied. They do not need to be modified unless users desire to use them to gather information or initialize or update their own variables.

3. Control routines that take care of bookkeeping, communicating with the monitor, calling the strategy and miscellaneous routines, saving and restoring variables, displaying the game, providing random numbers, etc. Participants should not need to change these routines, or even to understand how they work.

Note that routine names are written in upper-case (e.g., BID) in this document, but may need to be lower-case in actual programs.

In more detail, the user routines are as follows:

STRATEGY ROUTINES

BID	For a buyer, called at the start of each bid-offer step to decide on a bid value, or to pass.
OFFER	For a seller, called at the start of each bid-offer step to decide on an offer value, or to pass.
BUY	For a buyer, called at the start of each buy-sell step to decide whether to request to buy.
SELL	For a seller, called at the start of each buy-sell step to decide whether to request to sell.

Note that a player program can play a buyer or a seller, or may be able to play either. If only one of the roles is chosen, then only two of the four strategy routines needs to be developed.

However, in the tournament, players that can only play one role will be selected for fewer games, and thus have less opportunity for profit.

More details about the return values of these routines is given in the last section of this document.

MISCELLANEOUS ROUTINES

GBEGIN Called once at the beginning of each game.

GEND Called once at the end of each game.

RBEGIN Called at the beginning of each round.

REND Called at the end of each round.

PBEGIN Called at the beginning of each period.

PEND Called at the end of each period.

BOEND Called at the end of each bid-offer step, after all players' bids and offers have been processed.

BSEND Called at the end of each buy-sell step, after all players' buy and sell requests have been processed.

3.2 Variables in the user routines

The variables and arrays described below are available in all the user routines (strategy routines and miscellaneous routines). They provide all the information available to the player, on which bid/offer/buy/sell decisions must be made. The variable names and meanings are almost the same in all languages, and are defined here in a language-independent way. See the comments in the appropriate skeleton programs for all language-dependent details.

None of the variables described should be altered by the user routines themselves; doing so will not have any useful effect.

All variables are integers. Prices are represented by integers in the range 1—8000. It is of no consequence whether these values are regarded as dollars, cents, or some other unit.

Some variables are one-dimensional arrays of fixed length. The length is given in square brackets. Thus, for example, token[4] represents a 4-element array. How this is declared and referenced is language-dependent. In general array indices or subscripts run upwards from 1, not from 0.

It is convenient to divide the variables into three classes, according to whether they are public (the same for all players) or private, and whether they are constant or varying:

PUBLIC CONSTANTS

These variables are unchanging throughout a game (the same in all periods and steps), and are known to all players:

nplayers The total number of players in the game, including yourself.

nbuyers The number of buyers in the game. This will never exceed 20.

nsellers The number of sellers in the game. This will never exceed 20. Note that nbuyers+nsellers = nplayers.

bnumber[20] A player-number for each buyer, in the range 1—9999. There is one entry for each buyer, 1 to nbuyers, and the remaining entries are zero. These numbers are intended to give a unique identification to each program and participant across all games, so that participants in the pre-tournament Santa Fe Token Exchange games can determine who they have and have not played against. A directory will be available. A value of 9999 is used to mean a human player. Some values may be reported as 0 to mean anonymous; all players will be anonymous in the actual tournament. Do not confuse player-numbers with the player id's (described below) that are used to identify the players in a particular game.

snumber[20] A player-number for each seller. Like bnumber[20].

nrounds The maximum number of rounds in the game. This will never exceed 20.

nperiods The maximum number of periods per round. This will never exceed 5.

ntimes The number of time units in each period. This will never exceed 400. At each time there is first a bid-offer step and then a buy-sell step. A period will not be terminated before all 'ntimes' time units have been played if it is possible for any pair of players to trade with mutual profit. Early termination will not necessarily occur even when no more mutual profit is possible; see the description of the monitor's 'deadsteps' parameter.

minprice The minimum price value allowed for any bid or offer.

maxprice The maximum price value allowed for any bid or offer. Note that 1 <= minprice <= maxprice <= 8000.

gameid An arbitrary unique integer in the range 1 - 9999 that identifies this particular game. The gameid is useful for referring to particular games and for creating unique filenames, etc. It will normally increase from game to game.

gametype A 4 (decimal) digit number conveying the values for RAN1,..., RAN4. The RAN(i) values govern the random generation of tokens as described in rule 25 of chapter 7. The RAN(i) values are given by

$$RAN(i) = 3^{k(i)} - 1 \quad i=1,...,4$$

where k(i) is the i'th digit of gametype, counting from the left), and where $3^{k(i)}$ denotes 3 to the power k(i). Thus, for example, gametype = 1236 means

$$RAN1 = 3^1 - 1 = 2$$
$$RAN2 = 3^2 - 1 = 8$$
$$RAN3 = 3^3 - 1 = 28$$
$$RAN4 = 3^6 - 1 = 728$$

The values for the digits k(i) are restricted to (0,1,...,8). Values of gametype less than 1000 are taken as having leading zeroes. gametype = 0 implies that the token generation method is not revealed.

The upper limits on nbuyers, nsellers, nrounds, nperiods, and ntimes specified above may be relevant if you want to construct your own arrays to record historical or strategic information about other players or the game so far. In fact the upper limits will rarely be approached, and you can set lower limits of your own if necessary; see the skeleton programs for details. If the parameters of a particular game exceed your own limits, your program will automatically refuse to play in it. Note also that no token value is allowed to exceed 8000 (due to technicalities of the software design). If the RAN(i) values are set sufficiently high, this limitation will cause an upward truncation in the token distribution at 8000. However rule 25 in chapter 7 guarantees that in the tournament RAN(i) values will be chosen so that such truncation will never occur.

PUBLIC VARIABLES

These variables change with the progress of the game and are known to all players. All are set to 0 before the start of the game and (except r and p) before the start of each new period (i.e. before the call to PBEGIN). The "unknown" values referred to below can only occur if a player makes a 'late' response (see 'late' below).

r The number of the current round, r = 1, 2, ..., nrounds.

p The number of the current period within the current round, p = 1, 2, ..., nperiods.

t The current time within the current period, t = 1, 2, ..., ntimes.

cbid The current bid value if any, or zero if none. In a SELL routine this is the value you accept in a sell request. In a BID routine this is the value to beat; a new bid must be above this value. In a bid-offer step the highest bid (if any) determines the new cbid value before the call to BOEND. The current bid is removed (cbid=0) whenever a trade occurs, before the call to BSEND.

coffer The current offer value if any, or zero if none. In a BUY routine this is the value you accept in a buy request. In an OFFER routine this is the value to beat; a new offer must be below this value. In a bid-offer step the lowest offer (if any) determines the new coffer value before the call to BOEND. The current offer is removed (coffer=0) whenever a trade occurs, before the call to BSEND.

bidder The id of the holder of the current bid, or zero if none. Between 1 and nbuyers when cbid>0, always 0 when cbid=0.

offerer The id of the holder of the current offer, or zero if none. Between 1 and nsellers when coffer>0, always 0 when coffer=0.

nbids The number of new bids made by buyers in the last bid-offer step, or 0 if none or unknown. Updated before each BOEND call.

noffers The number of new offers made by sellers in the last bid-offer step, or 0 if none or unknown. Updated before each BOEND call.

bids[20] The actual bids made by each buyer in the last bid-offer step, or zero if none or unknown. There is one entry for each buyer, 1 to nbuyers, and the remaining entries are zero. The whole array is reset to reflect the new bids before each BOEND call.

offers[20] The actual offers made by each seller in the last bid-offer step, or zero if none or unknown. There is one entry for each seller, 1 to nsellers, and the remaining

entries are zero. The whole array is reset to reflect the new bids before each BOEND call.

bstype A code telling what happened in the last buy-sell step: 0 if no trade occurred, 1 if a buy request was accepted, 2 if a sell request was accepted, -1 if unknown because of a 'late' condition. This is set before each BSEND call. The next three variables provide more detail about the transaction if bstype>0. They are set at the same time as bstype. They are all set to zero if bstype<=0.

price The price of the transaction.

buyer The id of the buyer involved in the transaction. If bstype=1 this is a buyer whose 'buy' request was accepted. If bstype=2 this is the buyer whose current bid was accepted by 'seller'.

seller The id of the seller involved in the transaction. If bstype=2 this is a seller whose 'sell' request was accepted. If bstype=1 this is the seller whose current offer was accepted by 'buyer'.

btrades[20] A summary array giving the number of trades that each buyer has made so far in this period. There is one entry for each buyer, 1 to nbuyers, and the remaining entries are zero. A trade for a buyer is an accepted buy request or the acceptance by a seller of that buyer's current bid; in either case one token is consumed. Updated before each call to BSEND.

strades[20] A summary array giving the number of trades that each seller has made so far in this period. There is one entry for each seller, 1 to nsellers, and the remaining entries are zero. A trade for a seller is an accepted sell request or the acceptance by a buyer of that seller's current offer; in either case one token is consumed. Updated before each call to BSEND.

ntrades The total number of trades made by all players so far in this period.

prices[80] The price of every trade that has occurred so far in this period. There is one entry for each trade, 1 to ntrades, and the remaining entries are zero. The value is negative (multiplied by -1) if the trade was made by you.

lasttime The time (value of t) at which the most recent trade occurred in this period, or 0 if no trade has occurred yet in this period.

PRIVATE VARIABLES

Most of these variables change with the progress or the game and are in general different for each player. All are set to 0 before the start of the game. mytrades, mylasttime, pprofit, nobidoff, bo, nobuysell, bs, and late are reset to 0 at the start of every period and round. In addition at the start of each round, rprofit, tradelist[5], and profitlist[5] are reset to zero, and ntokens and tokens[4] are set to their new values.

id Your own identification number, between 1 and nbuyers if you are a buyer, and between 1 and nsellers if you are a seller. When the id number of another player is given, it is always clear from context whether it is that of a buyer or that of a seller.

role 1 if you are a buyer, 2 if you are a seller.

timeout The number of seconds of elapsed (wall) time that you have per step. 9999 implies infinity; the monitor will wait for ever. 0 means a relatively short time, normally used for pipe-based players, that depends on the monitor's 'timefactors' parameters.

ntokens The number of tokens that you have available to buy or sell. This is fixed for a given round, but may change from round to round. It will never exceed 4, and in the tournament it will be the same for all players.

token[4] The redemption values of your tokens, in token[1], ..., token[ntokens]. The redemption values are given in decreasing order for a buyer and in increasing order for a seller, and are assumed to be used in that order, which maximizes any profits. Any unused array elements are set to 0.

mytrades The number of trades (0 - ntokens) that you have made so far in this period. This is equal to the appropriate entry in btrades or strades. If mytrades<ntokens your next available token value is token[mytrades+1]. If mytrades=ntokens you have used all your tokens and cannot make any further bid, offer, buy, or sell requests.

mylasttime The time (value of t) at which your most recent trade occurred in this period, or 0 if you have not yet made any trades in this period.

pprofit Your total profit so far in this period of the game, or 0 if you have not yet made any trades in this period.

rprofit Your total profit so far in this round of the game, or 0 if you have not yet made any trades in this round.

gprofit Your total profit so far in this game, or 0 if you have not yet made any trades.

nobidoff 0 if you are allowed to make a bid or offer, 1 if you can't because you have no tokens left. -1 if unknown because of a 'late' condition. Reset before each call to BID or OFFER.

bo A code that tells you the outcome of your request in the last bid-offer step. This is updated before each call to BOEND. The meaning of the codes is as follows:

- 0 You didn't make a bid or offer, and you don't still hold the current one.
- 1 You didn't make a bid or offer, but your previous bid or offer is still the current one.
- 2 Your bid or offer was chosen and is now the current one.
- 3 Your bid or offer was bettered by another player.
- 4 Your bid or offer was equal to that of at least one other player, and you lost the random tie-break.
- -1 Your bid or offer was unacceptable. This could be because it was outside the allowed range (minprice to maxprice), or because it was a bid that wasn't above the current bid, or an offer that wasn't below the current offer, or because you have no tokens left to trade.

-2 A previous response to a bid-offer or buy-sell step was late and was ignored. See 'late' below.

Note that your new bid or offer is reported to the other players in cases 2-4, but not in case -1 or -2.

nobuysell 0 if you are allowed to make a buy or sell request, non-zero if not. -1 if unknown because of a 'late' condition. This is reset before each call to BUY or SELL. When non-zero the value is between 1 and 7, given by the sum of whichever of the following apply:

1 No tokens left to trade.

2 No current offer/bid to accept.

4 You don't hold the current bid or offer.

bs A code that tells you the outcome of your request in the last buy-sell step. This is reset after each buy-sell step. The meaning of the codes is as follows:

0 You didn't make a buy or sell request.

1 Your buy or sell request was accepted and you made a trade.

2 Your buy or sell request was rejected because both the current bidder and the current offerer asked to buy/sell, and you lost the random tie break (losing is usually better than winning here).

3 Your buy or sell request was rejected because another buyer or seller (the same type as you) also made a valid request, and won the random tie-break.

-1 Your buy or sell request was unacceptable. This could be because there was no corresponding current offer or bid, or because you had no tokens left to trade, or because you didn't hold the current bid or offer. This should not occur if you never try to buy or sell when nobuysell is non-zero.

-2 A previous response to a bid-offer or buy-sell step was late and was ignored. See 'late' below.

Note that only successful buy/sell requests (bs=1) are reported to the other players.

Note also that you may have made a trade even if bs is not equal to 1, because your current bid or offer may have been accepted by another player. Check whether 'buyer' or 'seller' (as appropriate) is equal to your 'id' to know for sure.

late The number of times in this period that you were late and failed to respond on time. If this is non-zero, be aware that the following variables may be incorrect in this period, even long after the late response, because information from the monitor may have been missed: btrades[20], strades[20], ntrades, prices, lasttime, mylasttime, pprofit. The following variables may be incorrect in this period and/or in future periods if you were ever late: rprofit, gprofit, profitlist[5], efficiency. The following variables may be temporarily incorrect, while either bo or bs is set to -2, but should recover correctly: nbids, noffers, bids[20], offers[20], bstype, price,

buyer, seller, nobidoff, nobuysell, t. Except for t this last group is set to well defined "unknown" values when a correct value is not known because of a late response; see the entries for the individual variables above. t may temporarily be smaller than its true value after a late response.

tradelist[5] The number of trades you made in each period of this round. There is one entry for each period, 1 to p, started so far in this round, and the remaining entries are 0. Updated after each trade.

profitlist[5] The profit you made in each period of this round. There is one entry for each period, 1 to p, started so far in this round, and the remaining entries are 0. Updated after each trade.

efficiency A measure of your overall performance in this game, available only in the GEND routine at the end of the game. The measure is a ratio (times 100) of your actual profit to your expected profit predicted by economic theory (computed at the midpoint competitive equilibrium price). 100 is average, higher values are better.

3.3 Working variables

If file-based communication is in use, each player program must be reloaded for each step, and all needed variables must be saved to a disk file and restored from it. The control routines do this automatically for all the variables discussed above, and can also save specified variables of your own. How this is done is language dependent; see the comments in the README file and skeleton programs for your chosen language.

If pipe-based or internet-based communication is used, the player programs are continuously loaded during a game, and the save/restore mechanism is not needed. You may create new variables as you please (including global or common ones), and expect them to retain their values according to the usual conventions of the language used.

Player programs may also create and use external files of their own; each player program will normally exist in its own sub-directory. Such files will be preserved from game to game.

3.4 Return values

In a BID or OFFER routine you must specify a bid or offer value, or none. In a BUY or SELL routine you must say if you want to buy or sell. The way that these return values from your strategy routines are passed back to the control routines is language-dependent. For details see the comments in the skeleton program for your language. Here we simply define the values concerned. All values are integers.

BID (buyer only) For a BID routine, return the value of the bid if you want to make one, or 0 if you don't. If you do make a bid it must be between minprice and maxprice inclusive, and must be higher than the current bid 'cbid' if there is one. You must return 0 if nobidoff is non-zero.

OFFER (seller only) For an OFFER routine, return the value of the offer if you want to make one, or 0 if you don't. If you do make an offer it must be between minprice and maxprice inclusive, and must be lower than the current offer 'coffer' if there is one. You must return 0 if nobidoff is non-zero.

BUY (buyer only) For a BUY routine, return 1 if you want to request a buy at the current offer price 'coffer', or 0 if you don't. You must return 0 if nobuysell is non-zero.

SELL (seller only) For a sell routine, return 1 if you want to request a sell at the current bid price 'cbid', or 0 if you don't. You must return 0 if nobuysell is non-zero.

CHAPTER 4: THE MONITOR

This chapter explains how to use the monitor, and how to interpret its output. The latter is useful not only to participants running a local monitor, but also to those using the Santa Fe Token Exchange (SFTE); the SFTE monitor mails its main (listfile) output back to each participant. Instructions for compiling the monitor are not given here; see the README file in the monitor source directory.

4.1 Starting the monitor

On an IBM PC two executable versions of the monitor are provided, called 'dmon' and 'gdmon'. The difference is that 'gdmon' has graphics built in, and turned on by default. 'dmon' is smaller and has no graphics capability. On other computers without graphics capability the monitor is normally called 'dmon'.

Before running the monitor you must set up the game file and players file in the same directory as the monitor. The versions provided, called 'game' and 'players' respectively, contain comments that document the format of these files, and some example entries.

The game file specifies the basic parameters for the game, including the number of rounds and periods etc, token generation parameters, and timing parameters. See the supplied 'game' file for details.

The players file specifies all local players (network players are controlled by parameters in the game file, if you have the full networking software). Each local player is specified by a directory name and a file name. The directory specification is relative to the monitor's own directory, so for example a directory 'playdir' means subdirectory playdir of the monitor's directory. It is recommended (for speed) that all players either be in such immediate subdirectories or be in the monitor's own directory, but other directory specifications (and, for the PC, drive specifications) are allowed. Examples are given in the supplied 'players' file. Note that all file-based players must have distinct pathnames, whereas pipe-based players may be specified more than once on Unix systems.

Each player may also be given one or more option flags in the players file. It is crucial that these be assigned correctly. See the description and examples in the supplied 'players' file.

Once the game file and players file are set up correctly, the monitor may be started with a command of the form

dmon [options] [listfile [logfile [gamefile [playerfile]]]]

It is essential that the current directory is set to the monitor's directory; starting the monitor from another directory will NOT work.

The options will be discussed later. The four optional file names have the following meanings and defaults:

listfile	Main output listing of the game. Defaults to stdout.
logfile	Initial setup report and error messages. Defaults to stdout.
gamefile	File specifying game parameters. Defaults to 'game'.
playerfile	File specifying local players. Defaults to 'players'.

Any of these filenames may be specified as - to mean the default. A + preceding logfile (e.g., "+log") implies append to the end. logfile may be specified as = to mean the same as listfile. If listfile or logfile contain any #'s they will be replaced by digits of the game number. E.g., out### becomes out284 for game 284 (or 1284).

Examples (for game 538):

dmon	everything goes to stdout
dmon out###	main report to out538, log to stdout
dmon out### log###	main report to out538, log to log538
dmon out### +dmonlog	main to out538, log appended to dmonlog
dmon out### =	everything to out538
dmon - -	newgame dummies everything to stdout; game and players files specified as 'newgame' and 'dummies'

Output redirection with > is not recommended for the monitor output. The monitor tries to write sensible things to its listfile and logfile depending on whether or not they go to files, and whether or not they go the same place; using redirection prevents this.

The following options may be specified:

-b Turns off terminal input. MUST be used to run the monitor in background/batch.

-g Turns on all graphics, if available. Currently only available on the PC. This is the default for 'gdmon'.

-g1 Restricts graphics to display only the supply and demand curves at the start of each round.

-g2 Restricts graphics to display only the transaction data (with supply and demand curves) at end of each period.

-ga Makes graphics automatic—it displays whatever you selected with the above options without asking you first.

-g0 Turns off graphics if on by default. Note: -g options may be concatenated. E.g., -g2a

-ppath Specifies PC directory where the .BGI and .CHR graphics files are located. E.g., -p\tc2. The default for the precompiled versions of the monitor is \tc. If the files are not found in the specified or default directory, they are searched for in the monitor's own directory. Note that the .BGI and .CHR files are provided on the floppy disk with dmon, but you probably already have copies on your hard disk if you have Turbo-C.

Example with options:

dmon -b out#### +log &

(The & is a Unix special character that makes the job run in the background.)

4.2 Runtime commands

If no player uses the terminal (no -t or -h flag in the players file), and there is no -b flag on the dmon command line, the following single-letter commands are available while the monitor is running:

A. While waiting for network players (inet version only):

s start the game now
a or q abort the game

B. During the main game:

t make this the last time step of this period (after next BS step).

p make this the last period of this round.

r make this the last round of the game.

e end the game; equivalent to 'rpt'.

q quit; like 'e', but suppresses mailback and any final graphic display.

k kill any players we're waiting for now.

c continue; force the next step now (may mark some players as late).

These commands do not echo and take effect immediately (though output may be backlogged, making an apparent delay). Any other character rings the bell. Use 'k' only when the monitor is stuck waiting for a player who is not responding.

4.3 VAX/VMS Notes

The monitor should work correctly on a VAX/VMS system. The treatment of file-based players is straightforward and should cause no problem, but may be very slow on a busy system. Pipe-based players are much more efficient, and thus faster, but more fragile. Each pipe-based player is started up as a sub-process of the monitor, and communicates with it via VMS "mailboxes." If something goes wrong it is possible for the player subprocess to continue running (in a wait state) even after the monitor exits or aborts. You can check for subprocesses with the command "SHOW PROCESS/SUB"—the player sub-processes are called PLAYER1, PLAYER2, etc. If you see any of these in the process tree, you can kill them with "STOP PLAYERn." The monitor will refuse to run a player if its chosen sub-process name already exists.

Note that you must have the TMPMBX and NETMBX process privileges, and a non-zero subprocess quota, to run the monitor. Most users will have these.

Be patient when starting a game. VAX/VMS takes a relatively long time to start up a subprocess when the system is busy.

4.4 Explanation of Monitor "Listfile" Output

The main output of the monitor is a concise summary of the game parameters, the players involved, and the actual play of the game. It is produced in the "listfile" output of the monitor, which defaults to stdout. In network games run at the Santa Fe Token Exchange, the listfile output is automatically e-mailed back to participants after successful game completion.

PARAMETER SECTION

The first section of the listfile output gives the parameters used in the game. For example:

DA game 1738 Sun Jun 4 19:52:45 1989

files: out1738	-	game players					
protocol:	5	monitor:			385	gametype:	1
nrounds:	2	nperiods:			2	ntimes:	50
minprice:	1	maxprice:			1000	ntokens:	4
ran1:	500	ran2:			0	ran3:	50
ran4:	5	deadsteps:			0	timeout:	30
btokens:	450	375	250	175			
stokens:	50	150	225	275			
timefactors:	1000	500	500	500	500	500	250

The heading gives the game number (gameid) and the date and time that the game began.

The 'files:' line gives the names of the files used for the listfile, logfile, gamefile, and playerfile, in that order. See the monitor's README file for details. The - in the above example means that the logfile output was sent to stdout.

'protocol' and 'monitor' give the version numbers of the message-passing protocol and monitor program respectively. They can normally be ignored.

The remaining parameter values are all specified in the gamefile. They are explained by comments in the distributed version of this file, called 'game'.

The internet version of the monitor (as used at the Santa Fe Token Exchange) adds several more parameters to the above list. The additional parameters are all concerned with controlling the waiting period for players, and the maximum and minimum number of players required for a game. They should not be important unless you have the internet version of the monitor, in which case you will find an explanation in the extended version of the 'game' file.

PLAYER SECTION

The second section of the listfile output lists the players involved in the game. For example:

id	num	flags	location	name/comment
B1	500	p	playdir/qp	Pipe C skeleton
B2	9999	isH	palmer@physics.phy.duke.edu	Richard Palmer
S1	502	p	fortran/pplayer	Fortran skeleton
S2	501	f	playdir/qf	File C skeleton

2 buyers, 2 sellers

The 'id' column gives the name used for each player, starting with B for buyers and S for sellers.

The 'num' column gives the player-number of each player; this should uniquely identify the player and the strategy. Participants are assigned a range of values to use. 9999 is used for human players. 0 means that a correct value is unknown or has not been assigned; participants should ask for a correct assignment rather than using 0 except temporarily.

The 'flags' show various status flags for each player. Most of these are specified or implied by the player descriptions in the playerfile; see the comments in the distributed version of this file called 'players'. The first column of flags describes the communication method, and may be f (file-based), p (pipe-based), a (pipe-based with descriptors passed in the argument string), or i (internet-based). The second column describes the timing option, and may be s (slow), w (wait), or blank (default). The third column describes the player/connection/output type, and may be blank (default), t (player requires terminal), h (human player), H (human player via internet), d (connected via DANI), or c (C-player via internet).

The 'location' gives the directory and filename (relative to the monitor's directory) for a local player, or the internet address of a network player. The internet address is not necessarily that at which the player is located, but is normally a suitable mailing address for the player. A numerical address enclosed in square brackets (e.g., fred@[192.12.12.2]) may be used with most mailers. An address or comment appearing in parentheses (e.g., joe@(local)) is NOT suitable for e-mail.

The 'name/comment' column gives the comments from the playerfile for local players, and the player or program name for network players. DANI and the human network interface both ask the player for his/her name, while the C internet-based player has a player name built into the program.

GAME SECTION—TOKEN DISTRIBUTION

The main game report consists of token-distribution headings, step-by-step lines, and summaries. The token-distribution headings appear at the start of each period. For example:

Round 1, Period 1					
token	B1	B2	S1	S2	Equilibrium
a	816+	816+	429+	427+	641 to 650
b	762+	765+	553+	553+	av: 645.5
c	658+	661+	641+	640+	trades: 6
d	555–	553–	650–	653–	

The tokens assigned to each player are labeled a, b, etc., with values as given in the table. Each value has a + or a - appended to it, depending on whether (+) or not (-) it would be traded if all trades occurred at the theoretical average equilibrium price. An = is used if the token value is equal to the average equilibrium price.

The 'Equilibrium' column shows the equilibrium price range and its average. There is often a finite range because the supply and demand curves usually have a vertical segment in common. The number of trades expected in equilibrium is also shown; this is normally equal to the number of + suffixes in both the buyer and the seller columns. '*no crossing*' appears below 'trades' if the supply and demand curves do not cross.

GAME SECTION—STEP-BY-STEP LINES

Each bid-offer and buy-sell step is summarized on a line. For example:

t	step	B1	B2	S1	S2	cbid	coff	price
1	BO	476*	469	717*	722	476	717	
	BS					476	717	
2	BO	549*	541	644*	645	549	644	
	BS	a<A		<A				644
3	BO	476*	469	717*	729	476	717	
	BS						476	717
4	BO	549	550*	667	654*	550	654	
	BS						550	654
5	BO	581	585*	623	623*	585	623	
	BS		a<A		a<A			623
...								
29	BO	639*		644*	D	639	644	
	BS				D	639	644	
30	BO	641*		642*	D	641	642	
	BS				D	641	642	
31	BO	642*		"	D	642	642=	
	BS	c>C		c>C	D			642
...								

The leftmost columns show the time t and whether the step is a bid-offer (BO) or buy-sell (BS) step. The rightmost columns show the current bid and current offer, if any, at the conclusion of each step, and the price of any transaction that occurred. An = sign or ! sign follows the current offer column if the current bid is equal to (=) or greater than (!) the current offer.

For bid-offer steps the central columns show the bids and offers made by each player. The current (highest) bid and current (lowest) offer are indicated by a * suffix. If there are no new current bids but one is left over from the previous bid-offer step it is shown by a double quote ("), and similarly for offers.

For buy-sell steps the central columns indicate requests to buy or sell by lower-case letters, and actual tokens sold by upper-case letters. The letter used identifies the token concerned. The direction of a transaction is indicated by the < or > symbols, which point like arrows from the accepted bid or offer to the acceptor; < means a buy request was accepted and > means a sell request was accepted. Thus for example in step 5 BS, both B2 and S2 made buy/sell requests and B2 'won' the random 50-50 tie break, so a 'buy' trade was executed at the current offer price. Note that it is in fact better to 'lose' such a random tie break unless the current bid is greater than or equal to the current offer.

Several other symbols may appear in the central columns in either bid-offer or buy-sell steps:

D means that the player is 'dead', for one of many reasons. Death is irreversible. The D symbol may also appear in the summaries described in the next section.

X means that the player's response was unacceptable, such as a bid below the current bid, or a buy request when there is no current offer.

N means that the player is non-responsive—no reply was received for this step.

L means that the player made a 'late' response, presumably to a previous step. A response is marked as late, and otherwise ignored, following any non-responsive (N) step.

? means that the response was invalid or unrecognized.

Note that the above symbols marking abnormal situations may be combined with some of the normal symbols. For example D" could occur in a bid-offer step if the holder of the current offer died and no other offer was made, and could be followed by D<B if a buyer subsequently accepted this offer, thus buying the 'dead' player's second (B) token.

A line is drawn across the output when no further mutual profit is possible. If the 'deadsteps' parameter is set to 0 this will cause the period to be ended, but if deadsteps > 0 further steps may appear after the line.

GAME SECTION—SUMMARIES

Summaries appear at the end of every period, and at the end of every round, and at the end of the game. The three types have basically the same appearance but apply to the preceding period, round, or game respectively. For example:

Round 1, summary					
	B1	B2	S1	S2	Market
Trades	6/6	6/6	7/6	5/6	24/24
Profit	667	697	559	532	2455
Eqlbrm	599	611	627	532	2470
Effncy	111%	114%	89%	84%	99%

The 'Trades' row shows the number of trades made by each player, out of the number predicted if all trades occurred at the theoretical equilibrium prices. The final 'Market' column shows the same information for the whole market (buyers plus sellers).

The 'Profit' row shows the actual profits made. The 'Eqlbrm' row shows the profits predicted if all trades occurred at the theoretical equilibrium prices. A . following an Eqlbrm value means 0.5 (e.g., 585. would mean 585.5).

The 'Effncy' row shows the 'Profit' row as a percentage of the 'Eqlbrm' row. The 'Effncy' may be shown as 0/0 (if Profit and Eqlbrm are both 0), or as 'inf' (if Eqlbrm is 0 but Profit is not), or as 'huge' (if greater than 9999%).

4.5 Explanation of Monitor "Logfile" Output

The monitor's "logfile" output is used to display the startup phase of the game, and to display error messages. It may be sent to the same file as the listfile output (in which case error messages are interpolated into the game report), or to a different file. Depending on whether the files are the same or different, and whether the output is to a named file or to stdout (the default), the logfile output also shows more or less information on the progress of the game.

STARTUP PHASE

During the startup phase the logfile shows one line for each player that the monitor tries to start. For example:

```
dat@192.12.12.1:      connected—S1 C skeleton player
palmer@192.12.12.1:   connected—B1 Richard Palmer
fortran/fplayer:      verified—S2
playdir/refuser:      started—B2
fortran/pplayer:      started—S3
playdir/pq:           non-existent or non-readable
playdir/qf:           verified—B3
```

This example shows that two network players connected, and then the monitor tried to start five local players, though one of these (playdir/pq) didn't actually exist. The numerical address given for the network players is the actual internet address from which they connected. For the local players 'started' implies that a pipe-based player was successfully started, while 'verified' implies that a file-based player was checked for existence and accessibility. Players are tentatively assigned identification numbers (S1, B1, etc), though they may be renumbered later.

At a second stage of the startup phase players are told the game parameters and asked if they want to play. Any refusals produce messages in the logfile such as:

```
B2 refused to play    — nrounds unacceptable
                      — maxtokens unacceptable
```

This will then cause renumbering of the remaining players before the main listing of players that appears in the listfile.

ERROR MESSAGES

Error messages from the monitor are intended to be self-explanatory. Some typical ones are:

```
** error reading 'ntimes' from 'game': non-digit found
** error reading 'ntokens' from 'game': value too large
** S1  died in INIT2 0 0 0—fatal error
** B1  died in BIDOFF 1 1 1—quit
** S2  died in BUYSELL 1 2 8—connection broken
** B2  died in BIDOFF 2 2 1—killed by operator
*** no active buyers left—game ended
```

Many of the messages show the step in which the error occurred, such as the 'BUYSELL 1 2 8' above, which means the buy-sell step at time 8 of period 2 of round 1. There are many possible reasons for a player dying, and the difference between reasons like 'read error', 'connection broken', 'write error' and 'unresponsive' should only concern an expert; in each case the player program stopped responding properly. The reason 'fatal error' means that the player program itself detected an error and called the 'error' subroutine. An interrupt can also cause this error. The reason 'quit' means that the player intentionally quit the game; this is normally only used by the human interface.

LOG INFORMATION

If the logfile and listfile are not identical, the logfile shows the progress of the game as it runs. More information is shown if the logfile is defaulted to stdout than if it is sent to a named file. In the stdout case an example is:

```
DA game 1745   Mon Jun  5 17:06:22 1989

files: out1745              - game  players
Initialization-1
Initialization-2
Round 1
     period 1
*-*<*-*-*>*-*-*<*-*-*<*-*-*-*-*-*-*<*-*-*-*-*-*-*<|
     period 2
*-*<*-*<*-*-*<*-*-*<*-*-*-*-*-*-*<*-*-*-*-*-*-*<|
Round 2
     period 1
*-*-*<*-*-*>*-*-*<*-*-*<*-*-*-*-*-*-*-*-*<*-*-*-*-*-*-*-*-*-*<|
     period 2
*-*-*<*-*-*<*-*-*>*-*-*<*-*-*-*-*-*-*-*-*<*-*-*-*-*-*-*-*-*-*<|
End of game 1745—output is in out1745
```

The heading and 'files:' line are identical to those in the listfile. The 'Initialization', 'Round', and 'period' lines log the major divisions of the game.

The lines with asterisks are produced one symbol at a time if output is to stdout. One symbol is shown for each bid-offer and each buy-sell step. For a bid-offer step a * is shown if there were any bids or offers; otherwise a . is shown. For a buy-sell step a < is shown for a buy transaction, a > is shown for a sell transaction, and a - is shown if no transaction occurred. A | is shown when no more mutual profit can be made.

CHAPTER 5: THE PLAYER PROGRAMS

This chapter explains how to use the player programs, including the human interface. Compilation instructions are not included here; see the appropriate README files and makefiles.

5.1 Starting the Player Programs

INTERNET-BASED PLAYERS

To start an internet-based player (including a human/internet player), simply run the program. There are no options or arguments (unless you have the full networking software, in which case a hostname may be specified as an argument). The human/internet player will ask for your name (maximum 30 characters) and whether you want to be a buyer or a seller. If the program successfully connects to the monitor you will then receive one of four types of message from the monitor:

1. A description of the game schedule in a box of *'s, followed by the time at the monitor's location. This means that no game is presently starting; try again at the appropriate time.

2. A "Waiting for ..." message. This means that a game will start as soon as the condition is satisfied. Further waiting messages will be sent every 30 seconds until the game starts or is abandoned.

3. A "Game starting" message. This means that your connection satisfied the condition for a game, which is now starting.

4. A "Sorry, ..." message. This means that a game is about to start, but that you cannot join it for the reason given.

Note that all messages before the beginning of the game itself are sent to stderr, not stdout. This implies for example that the messages will come to the screen even if you redirect the main (stdout) output to a file, as may be appropriate for a game run in background. If you start up several players, with all but one in background, you will see multiple copies of the messages, one from each player.

It may be useful on a Unix system to copy the main (stdout) output to a file as well as sending it to the screen. Pipe the output into 'tee' to do this. E.g., :

```
di | tee output
```

Here 'di' is the name of the internet player, and 'output' is the name of the output file.

PIPE-BASED OR FILE-BASED PLAYERS

You do not normally run pipe-based or file-based player programs by themselves, but expect the monitor to do so. The "monitor" chapter of the documentation explains how this is done.

The human interface is used like the machine players except for the following:

a. The -h flag is required in the playerfile.

b. The monitor's listfile output must be sent to a file, not the screen. Specify the filename on the command line. E.g., "dmon outfile."

Using the human player via DANI is not recommended. Instead make an inetbased version of the human player itself (with DISPLAY turned on in define.h). If you DO use a human player via DANI, you must use the -a and -q flags if pipebased (-f and -q if filebased).

If in doubt as to whether the players are working correctly it may be useful to run them directly. A useful test is to send the player the 15 character message ...31.1234....0 (where each . represents one space); the response should be ...31.1234 (possibly with one extra leading space). For the pipe- based case the messages are simply sent to and from stdin and stdout. For the file-based case the messages must come from the file xxxx.in and are sent to xxxx.out, where xxxx is the player program's name. After performing this test in the pipe-based case you must abort the player program, or send it a nonsense message to make it abort.

5.2 Explanation of human player and C-player output

The skeleton players written in C can optionally display the game as it progresses. This is a compile-time option (see the README file and comments in the programs), and is mainly intended for use in the human player. It may also be useful with machine players, especially with internet-based communication. Note that using DANI is an alternative in this case however, and has a more compact output format. The only difference between the output of the human player and of a C-player with output display is the prompts and responses used with the human player. These are discussed separately below.

The output display consists of initial headings, the detailed play of the game, and various summaries.

INITIAL HEADINGS

The first part of the output gives basic game parameters and a list of player numbers. For example:

```
DA game 1760     Tue Jun  6        21:32:20 1989
_____________________________________________________

nrounds: 2       nperiods: 2       ntimes: 50       price range: 1-1000

2 buyers:        B1  501           B2     9999
2 sellers:       S1  502           S2        0
You are seller S1, otherwise known as YOU
```

The parameters and game-id (1760 in the example) should be self-explanatory. Note that the date and time is local to the player's location, which may be in a different time-zone than that of the monitor itself.

The lists of buyers and sellers give their player numbers, or 0 if unknown or anonymous. These numbers may be useful in identifying strategies you have and have not played against. 9999 is used for a human player.

MAIN GAME

The main game is displayed step by step. The format is less concise than DANI's, and takes 6 lines per time step. For example, showing three time steps:

```
--- round 1 ---------- period 2 ---------- time 6 ----------
cbid: (none)          coffer: (none)          tokens: 542 450 303 264
                B1/c  YOU/a  S1/b  S2/b
bettered        183*   173   378*  395
no trade

--- round 1 ---------- period 2 ---------- time 7 ----------
cbid: 183/B1          coffer: 378/S1          tokens: 542 450 303 264
                B1/c  YOU/a  S1/b  S2/b
winner          218    247*  330*  333
profit: 212             <     30<

--- round 1 ---------- period 2 ---------- time 8 ----------
cbid: (none)          coffer: (none)          tokens:  - 450 303 264
                B1/c  YOU/b  S1/c  S2/b
lost tie        183*   183   392   383*
no trade
```

The first line at each time gives the round, period, and time.

The second line gives the current bid and offer, if any, and the player's tokens. Note that the owners of the current bid and offer are given if applicable. Note also that tokens that have been traded are replaced by dashes (-).

The third line lists the player names (with "YOU" for the present player), with a lower-case letter showing which token they will trade next (a = first, b = second, etc.).

The fourth line shows the bids and offers in the bid-offer step, with an asterisk marking the current bid and current offer. A current bid or offer carried over from the previous step (in the absence of any new ones) would be shown with a # instead of a *. On the left is shown the disposition of your own bid or offer request if you made one. "winner" means you had the best (or equal best) bid or offer, and now hold the current bid or offer. "bettered" means that someone else had a better bid or offer. "lost tie" means that you had an equal best bid or offer, but lost the resulting random tie-break.

The fifth line shows the result of the buy-sell step. If a trade occurs it is shown by a pair of <'s for a buy or a pair of >'s for a sell. The price is shown under the current bid or offer that it derived from. On the left is shown any profit that you made, or "no trade" if no trade occurred, or blank if a trade occurred in which you were not involved. "lost draw" means that you lost a random tie break involving other buyers or sellers (the same as you) who also made buy or sell requests.

In either the bid-offer or the buy-sell line the left column can also show **illegal** or **late**. **illegal** means that the your response was illegal, such as a bid below the current bid, or a request to buy when there was no current offer. **late** means that you missed responding to one or more steps, so the actual response was to a previous step and was ignored. Note that if this occurs, subsequent information about the number of tokens traded (a, b, etc), and some summary information, may be incorrect because relevant information was missed. See the description of the variables in the skeleton programs for a more detailed discussion.

SUMMARIES

At the end of each period a short summary appears:

> You made 3 trades, making a total profit of 286 in this period
> Summary of trade prices (* = yours):
> 538* 545 521* 535 513* 509

The number of trades and the profit are just for the period that just ended.

At the end of each round the number of trades and profit is summarized for the current player for each period of the round. For example:

```
================== End of round 1 ==================
Summary:     period   trades   profit
                1        3      286
                2        3      293
              total      6      579
```

At the end of the whole game a final summary appears too:

```
================== End of game 1760 ================
Total profit: 1186
Efficiency: 109%
```

The total profit is the total reported by the monitor. If that differs from the amount computed by adding up the individual profits for each round, the message will look like:

Total profit: 1186 (1037 accounted for)

This can occur if the player was **late** and missed a transaction.

The 'Efficiency' is the performance figure of merit reported by the monitor; the ratio of actual profit to that expected if all trades occurred at the theoretical equilibrium price. Higher values represent better performance.

5.3 Prompts and Responses in the Human Player

The previous section describes the display output of the human player besides the prompts and responses. Normally the human player asks for your decision once in each bid-offer and buy-sell step.

There are two types of prompts:

1. Prompts ending in a ? expect a substantive reply, either a number or a 'y' for yes or 'n' for no.

2. Prompts ending in a > are merely informational, and expect only a carriage return in response.

At either of these types of prompt, various options or commands can be specified instead of the expected response. The prompt is then repeated.

BID-OFFER PROMPTS

In a bid-offer step the usual prompts are:

	bid [n]?	for a buyer
or	offer [n]?	for a seller

These invite you to type in a bid or offer value respectively. If you don't want to make a bid or offer you can type an n (none), or just press return. The '[n]' reminds you that the default action is 'none'.

Instead of one of the above prompts you may receive one of the informational prompts:

	nothing to trade	if you've traded all your tokens
or	unwise to bid	if the current bid is above your next token
or	unwise to offer	if the current offer is below your next token

The last two 'unwise' messages only appear if you set the 'u' option discussed below.

BUY-SELL PROMPTS

In a buy-sell step the usual ? prompts are:

	buy at xxx [n]?	for a buyer
or	sell at xxx [n]?	for a seller

Here xxx is the current offer or current bid price respectively. Your response should be y for yes or n for no. The default if you just press return is shown in square brackets (n for no in the example, but this can be reversed with the 'a' option).

Instead of one of the above prompts you may receive one of the following informational prompts:

	not bidder>	for a buyer not holding the current bid
or	not offerer>	for a seller not holding the current offer
or	nothing to buy>	for a buyer when there's no current offer
or	nothing to sell>	for a seller when there's no current bid
or	nothing to trade>	if you've traded all your tokens
or	unwise to buy>	for a buyer who'd make a loss by buying
or	unwise to sell>	for a seller who'd make a loss by selling

The last two 'unwise' messages only appear if you set the 'u' option discussed below.

OPTIONS AND COMMANDS

At any prompt you may set or reset any of the following options. To set them simply type their letter. To reset them type a - and then their letter. Options can be concatenated, and a - applies to all subsequent options. For example, u-aw would set u and reset a and w.

u prevent 'unwise' choices likely to lead to a loss. This enables the 'unwise to ...' prompts detailed above, and refuses to accept any bid or offer that would result in a loss if accepted.

a accept bids or offers by default on the buy-sell step. This makes y (yes) the default for the 'buy at ...' and 'sell at ...' prompts.

f fast. Removes the wait for a carriage return (or timeout) at all '>' prompts. The message is shown without the '>', but no response is needed or allowed. Note that this may lead to rapid scrolling of the game on the screen, and does not let you enter further options or commands until a ? prompt next occurs.

w wait. This turns off the automatic timeout and waits for ever for your response. On most systems the human player types one '.' every second (or a ':' in the last five seconds) and then defaults your response (as if you had just pressed return) after a certain time. If you turn this off with the w option you may exceed the time that the monitor allows for your response, which may lead to a **late** disposition or even to you being 'killed'.

P Pass. This causes every prompt (both ? and > types) to be defaulted for the rest of this period. Note that P is upper-case.

The P option is reset at the start of each period but the other options are not.

Besides the above options which may be toggled on or off, the following commands may be given to any prompt:

quit Quits the game, which then continues without you.

help Prints a summary of these options and commands.

? Tells you what you can do in response to the current prompt.

t Shows your token values and the current bid and offer values.

$ Shows the transaction prices so far in this period.

h Shows a 'history' summary of the last few steps. The output is described in more detail below.

5.4 History Display in the Human Player

The human player can display a compact summary of the previous N steps in the current period, where N is a compile-time option. Use 'h' in response to any prompt to obtain the history display. The output combines each bid-offer and buy-sell step into a single line. For example:

t	YOU	B2	S1	S2	price
1	400*	321	514	510*	
2	435*>	433	476*>	477	435
3	400*	338	514*	518	
4	430	436*<	479*<	484	479
5		339*	514*	527	
6	390*>	383	488	463*>	390
7	350*	336	514	510*	
8		399*	485	472*	
9	425*>	423	459	446*>	425
12 !	395	415*<	470	469*<	469
13	350*	330	514*	523	
14	380*	378	488*	497	
15 *	392	399*	470*	478	

The leftmost column shows the time t. This may be shown as a ? if unknown because of a 'late' condition. The time column may be followed by a !, calling attention to the fact that some steps were missed, or a *, meaning that the information on this line is incomplete. Late responses can cause either of these conditions, but * is also normal on the last line if only the bid-offer step has occurred so far.

The columns for each player show the bid or offer values, followed by a * marking the current bid or offer, and a < for a buy or a > for a sell. The actual transaction price is repeated in the last column. A # is used instead of a * for a current bid or offer that was carried forward from a previous bid-offer step.

CHAPTER 6: DANI

This chapter explains how to use the Double Auction Network Interface (DANI), and how to interpret its output. Compilation instructions are not given here; see the README file in DANI's source directory.

6.1 Communication Methods

DANI acts as an interface between a monitor on the internet—normally the Santa Fe Token Exchange (SFTE)—and a local player. DANI communicates with the monitor via sockets over the internet, and with the player program by one of the following methods:

pipe-based — DANI starts up the player program once (as a child process) and attaches pipes between it and the player's stdin and stdout.

arg-based — This is just like pipe-based except that the pipes are attached to special descriptors specified as arguments on the player's command line instead of using stdin and stdout. For C players only. Useful for players that need to use stdin and stdout themselves.

file-based — DANI starts up the player program for each step of the game, reading and writing files for communication.

user-based — DANI does not start up the player itself, but reads and writes messages to the player on its stdin and stdout. Used for players on remote machines (e.g., dial-up lines) that can talk to DANI directly.

There is considerable overlap between this chapter and chapter 5, because DANI's display and startup messages are similar to those of the internet- based C players.

6.2 Starting DANI

Before starting DANI (except to inquire about the current status at the SFTE), you must compile and link your player program as well as DANI itself. Then start DANI with a command of the form:

dani [options] [playername] [hostname]

The options are explained below. The other parameters are:

playername — the name of the player program to be started. This must be in the current working directory (though DANI needn't be). 'playername' cannot be specified if you use the -u option. Otherwise if you omit it, DANI just inquires about the current status at the Santa Fe Token Exchange or elsewhere.

hostname — the name of the host to connect to on the internet. This only works if you have the full networking software, which is not normally distributed. Otherwise the host is taken as **sfi.santafe.edu**, for the SFTE. 'hostname' must be the second argument (after playername) unless the -u option is used. See also the -h option below.

OPTIONS

-b Insist on being a buyer.

-s Insist on being a seller.

If neither -b nor -s is specified the monitor will assign your role as buyer or seller.

-p Use pipe-based communication.

-a Use arg-based communication.

-f Use file-based communication.

-u Use user-based communication. This inhibits display or log output unless redirected with -o.

If none of -p, -a, -f, or -u is specified the default depends on a choice made when dani was compiled.

-n name Specifies the name of the player. If this is not supplied DANI will ask for it with "Enter your name:."

-h hostname An alternative way of specifying a hostname if you have the full networking software.

-d Turns on display of the game as it progresses.

-q Turns off display of the game ('quiet').

If neither -d nor -q is specified the default depends on a choice made when dani was compiled.

-o output Sends the display or log output to file 'output'. Defaults to stdout. Redirection with > may also be used, but -o may be useful if the player program also produces stdout output (in which case you must use -a or -f). -o MUST be used if you want to produce display or log output with user-based (-u) communication.

-l Log messages. For debugging only. Implies -q. Messages generated by DANI itself are prefixed by a *.

-k Don't send KILLED messages to player. For debugging in the filebased case—to prevent the player deleting its files.

After starting and testing the player, DANI will ask for your name (maximum 30 characters) unless you already specified with the -n option, or are only inquiring about the status (no playername given). Then, if it successfully connects to the monitor, you will then receive one of four types of message from the monitor:

1. A description of the game schedule in a box of *'s, followed by the time at the monitor's location. This means that no game is presently starting; try again at the appropriate time.

2. A "Waiting for ..." message. This means that a game will start as soon as the condition is satisfied. Further waiting messages will be sent every 30 seconds until the game starts or is abandoned.

3. A "Game starting" message. This means that your connection satisfied the condition for a game, which is now starting.

4. A "Sorry, ..." message. This means that a game is about to start, but that you cannot join it for the reason given.

Note that all messages before the beginning of the game itself are sent to stderr, not stdout or the -o file.

6.3 VAX/VMS Notes

DANI should work correctly on a VAX/VMS system as long as the Wollongong WIN/TCP networking software is available. The treatment of file-based players is straightforward and should cause no problem, but may be very slow on a busy system. Pipe-based players are much more efficient, and thus faster, but more fragile. The player is started up as a sub-process of DANI, and communicates with it via VMS "mailboxes." If something goes wrong it is possible for the player subprocess to continue running (in a wait state) even after DANI exits or aborts. You can check for subprocesses with the command "SHOW PROCESS/SUB"—the player sub-process is always called PLAYER. If you see PLAYER in the process tree, you can kill it with "STOP PLAYER." DANI will refuse to run again if PLAYER already exists.

Note that you must have the TMPMBX and NETMBX process privileges, and a non-zero subprocess quota, to run DANI. Most users will have these.

Be patient when starting up DANI. VaX/VMS takes a relatively long time to start up a subprocess when the system is busy.

6.4 Explanation of DANI's display output

DANI is able to display the progress of a game as it interfaces between the monitor and a player. This is enabled by the -d option and disabled by the -q option; it may or may not be the default, as fixed before compilation.

DANI's display output consists of initial headings, the main game, and various summaries. The initial headings and summaries are identical to those of the internet-based C players, but the main game display is different.

INITIAL HEADINGS

The first part of the output gives basic game parameters and a list of player numbers. For example:

```
DA game 1760  Tue Jun  6 21:32:20 1989
-------------------------------------------------------

nrounds: 2          nperiods: 2  ntimes: 50      price range: 1-1000

2 buyers:           B1           501  B2         9999
2 sellers:          S1           502  S2         0
You are seller S1, otherwise known as YOU
```

The parameters and game-id (1760 in the example) should be self-explanatory. Note that the date and time is local to DANI's location, which may be in a different time-zone than the monitor itself.

The lists of buyers and sellers give their player numbers, or 0 if unknown or anonymous. These numbers may be useful in identifying strategies you have and have not played against. 9999 is used for a human player.

MAIN GAME

Each period of a game consists of headings and then one line for each bid-offer and each buy-sell step. For example:

```
------- round 1 --------------- period 1 ----------------- tokens: 323 457 506 537 ------------------
t   profit            B1      B2     |  YOU      S2           price
1   winner            392*    380    |  602*     614
    no                               |  .
2   winner            455     465*   |  538*     543
    yes       215             <A     |  .<A                   538
3   winner            392     405*   |  602*     616
    no                               |  .
4   bettered          465*    450    |  558      545*
                       <A            |  .         <A          545
5   bettered          392     400*   |  602      595*
                                     |  .
6   bettered          464     465*   |  553      546*
                                     |  .
7   winner            490*    480    |  521*     525
    no        64       <B            |  .<B                   521
8   winner            392     400*   |  602*     609
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
16  lost tie          507     510*   |  518      518*
                                     |  .
17  bettered          511     512*   |  514      513*
                                     |  .
18  winner            513*    513    |  512*     512
    yes       7        >C            |  .>C                   513
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
-------------------------------------end of period 1 of round 1---------------------------------------
```

The heading line shows the round and period number, and the assigned token values. Note that these token values are usually scrolled off the screen as the game progresses, and may be worth writing down.

Bid-offer steps are labeled on the left with the time; buy-sell steps are unnumbered. A vertical bar (|) always appears between the buyers and the sellers, and a period (.) appears in the column for the current player ("YOU") in each buy-sell step; these are useful after the headings have scrolled off the screen.

Each bid and offer is shown in a bid-offer step, with asterisks (*) marking the current bid and offer. If there is no new bid, but a previous current bid is still outstanding, it is shown in the bidder's column with a # instead of a *, and similarly for offers.

To the left of the bids in a bid-offer step is shown the disposition of the current player's bid/offer/none request. 'winner' means that the bid or offer was the best (or equal best) and thus became the current bid or offer. 'bettered' means that a better bid or offer was received. 'lost tie' means that the bid or offer was equal best, but the current player lost the resulting random tie-break.

In a buy-sell step a transaction is shown by a pair of <'s for a buy, or a pair of >'s for a sell. The token traded for each player is shown by an upper-case letter (A = first, B = second, etc.). The transaction price is given in the rightmost column. This price column is omitted if there are the maximum number of players; the price is easily deduced from the preceding current bid or offer.

To the left of the transaction information in a buy-sell step is shown the disposition of the current player's buy/sell/none request. 'yes' means that a buy or sell request was made and carried out. '(yes)' means that a buy or sell request was made, but was not accepted because another player also made a valid request and won the random tie-break. This is usually better than 'yes' if you are the current bidder or offerer, because the trade occurs at YOUR price. 'no' means that a buy or sell request was permitted but was not made. The space is left blank if no buy or sell request was permitted.

If any profit was made it is shown in the profit column in the appropriate buy-sell step. Note that this can occur with a disposition of 'yes', '(yes)', 'no', and even blank.

The dispositions '**bad**' and '**late**' can appear in either step. '**bad**' means that the current player's response was illegal, such as a bid below the current bid, or a request to buy when there was no current offer. '**late**' means that the player missed responding to one or more steps, so the actual response was to a previous step and was ignored. Note that if this occurs, subsequent information about the token number traded (A, B, etc), and summary information described below, may be incorrect because relevant information was missed. See the description of the variables in the skeleton programs for a more detailed discussion.

SUMMARIES

At the end of each period a short summary appears:

```
You made 3 trades, making a total profit of 286 in this period
Summary of trade prices (* = yours):
538*  545   521*  535   513*  509
```

The number of trades and the profit are just for the period that just ended.

At the end of each round the number of trades and profit is summarized for the current player for each period of the round. For example:

```
================== End of round 1 ==================
Summary:      period    trades    profit
                 1         3        286
                 2         3        293
              total        6        579
```

At the end of the whole game a final summary appears too:

```
================== End of game 1760 ================
Total profit: 1186
Efficiency: 109%
```

The total profit is the total reported by the monitor. If that differs from the amount computed by adding up the individual profits for each round, the message will look like:

```
Total profit: 1186  (1037 accounted for)
```

This can occur if the player was **late** and missed a transaction.

The 'Efficiency' is the performance figure of merit reported by the monitor; the ratio of actual profit to that expected if all trades occurred at the theoretical equilibrium price. Higher values represent better performance.

CHAPTER 7: TOURNAMENT RULES AND REGISTRATION

This chapter specifies the rules, parameter settings and entry procedures for the actual Double Auction tournament to be held in March 1990. The rules of the Double Auction itself are given in the Introduction chapter; this specifies the rules relating to tournament entries and the distribution of the $10,000 prize money. The structure of the tournament is also discussed. An entry form appears at the end of the chapter.

7.1 Tournament Rules

1. The deadline for entries is March 1, 1990. Entries must be received at the Santa Fe Institute by 5 PM (MST) on this date to be eligible for inclusion in the tournament. If an entry conforms to all the rules listed below it will be accepted as a PLAYER in the DA tournament. If an entry has been submitted before the deadline and is found to violate one of the rules listed below, it will be returned to the entrant for revision and conditional acceptance provided the revised strategy is received before the deadline. Entrants will have at most one chance to submit strategies that conform to the rules. In the sequel we will refer to an accepted computer program (source and associated materials as described below) as a "player," and "entrant" as the human(s) who submitted it.

2. A maximum of 100 players will be accepted. Of these, 70 will be accepted on a first-come, first-served basis starting on September 1, 1989 until the entry deadline of March 1, 1990. The remaining 30 slots have been reserved for selection by the tournament organizers to guarantee sufficient diversity in the types of strategies submitted. If the organizers do not use all 30 slots, those that remain will be allocated on a first-come, first-served basis as described above. A list of current entrants' names and E-mail addresses is available by ftp in the "entries" file, or by request from the organizers. Entrants who wish to remain anonymous throughout the tournament can have their names appear as "anonymous" in the entries file.

3. A valid entry consists of

   A. source code for the user routines in one of the skeleton programs, subject to additional restrictions listed below, and

   B. a $10.00 entry fee payable to Santa Fe Institute, and

   C. a signed, completed registration form (at the end of this chapter).

4. The user routines referred to in 3A are the 4 Strategy Routines and 8 optional Miscellaneous Routines listed in the "skeleton" chapter of the documentation. They must be accompanied by appropriate declarations and/or definitions of the variables and parameters, as required by the language used.

5. The routines must be written in C, Fortran or Pascal and must be compatible with the skeleton programs as provided in these languages.

6. As an alternative to 3A, 4, and 5, users may submit source for a complete self-contained player program. This must:

   A. Conform to the message-passing protocol described in the "messages" chapter of the documentation.

B. Be able to communicate using the PIPEBASED method with the monitor on a Sun-4 system; this mainly requires that messages be read from stdin and written to stdout.

C. Be supplied in source form in C, Fortran, or Pascal. Other languages may be considered, but only by prior arrangement with the organizers.

7. All players must be sufficiently documented by comments in the source, and optionally by additional written (or ascii text) materials. The documentation should include at least the following:

   A. The name, affiliation, and address of each author. These MUST appear in comments in the source, as well as on the entry form.

   B. A general intuitive description of what the strategy does.

   C. Definition of all significant variables and parameters.

   D. References to any specific algorithms or numerical methods used.

   E. Description of the content and use of any external files needed or created by the program.

   F. Notes on any special requirements for compiling, linking, or running the program.

   We reserve the right to reject programs which we deem to be poorly documented or unnecessarily obscure. In such cases the authors will be invited to make improvements within a reasonable time before any final rejection.

8. Participants' programs may create and use external files, which will be preserved from game to game. Each player program will have its own directory, and may only access files within that directory. Additional files may be submitted with a tournament entry, but must be adequately documented.

9. Players must be runnable on a Sun-4 running SunOS 4.0. This means especially that only commonly available library routines should be used. Players will be checked for runnability upon receipt, and the organizers will make reasonable efforts to make any minor alterations necessary to suit local conditions. Ultimately however, we reserve the right to reject programs that we cannot run. Authors should contact the organizers if in doubt about the availability of library routines or language features at Santa Fe.

10. There are no explicit CPU time, memory, or disk storage limits for player programs. However if a program is found to use greatly excessive amounts of any of the above resources (relative to what is used by other players), the organizers reserve the right to request the entrant to modify the program so that its requirements are more in line with other players or, as a last resort, to disqualify the program.

11. Submitted players cannot be withdrawn or revised once submitted, except to satisfy the requirements these rules. Major changes in the player program will be prohibited once submitted.

12. Although joint entries are permissible, at most one player will be allowed per entrant. Duplicate or plagiarized player programs will be disqualified; all submissions must be distinct.

13. Players are only allowed to use information

    A. explicitly passed to them by the monitor, or

    B. stored in their private files (optional).

    Different players may not communicate directly, and may not share files. Any attempt to violate these rules (in letter or spirit) will be grounds for disqualification.

14. Both tournament entrants and non-entrants may use the Santa Fe Token Exchange (SFTE) for practice games if their hardware and software permits it. No guarantee of availability on a particular system or at a particular time is made. Traders on the SFTE may be either human or robot traders. There will be no cash profits paid for trading on the SFTE, and the tournament will be run independently of the outcome of any SFTE games.

15. Traders playing as local (non-network) players on the SFTE will be selected from those developed at SFI. Different selections may be made at different times of day. Participants are invited to submit their programs for this purpose, though no guarantee of using them is made. The organizers guarantee the confidentiality of programs submitted for local SFTE use (by using only the binary executable version of the program in an inaccessable file).

16. Prize money totaling approximately $10,000 will be paid to participating entrants in proportion to the total trading profits earned by their player programs in all tournament games in which they play (see description of parameter settings below).

17. In order to encourage entrants to develop trading strategies that perform well in a wide variety of environments, the parameter settings in the Double Auction tournament games will be systematically altered. In the tournament games the base token values (see the 'game' file) will normally be set to zero, so the relevant parameters consist of the following 10 variables (for more detailed definitions, see chapters 1 and 3, and 'game' file):

NTOKENS	number of tokens assigned to each trader
NBUYERS	number of buyers
NSELLERS	number of sellers
NROUNDS	number of rounds in DA game
NPERIODS	number of periods per round
NTIMES	number of bid/offer and buy/sell steps per period
RAN1	uniform random token generator upper bound 1
RAN2	uniform random token generator upper bound 2
RAN3	uniform random token generator upper bound 3
RAN4	uniform random token generator upper bound 4

    The meanings of RAN1, ..., RAN4 are discussed in detail in rule 25 below.

18. Games in the Double Auction Tournament will be selected from a fixed number of *environments*, E(i), i=1,...,I. Each environment E(i) is a complete specification of the 10 DA game parameters listed in 17. All the above parameters are passed to each player program at the start of each DA game (except that RAN1-4 and the token generation method may be withheld in a few environments). This guarantees that players have common knowledge about the environment, a game-theoretic consideration.

19. The specific environments used will be chosen by the organizers in advance of the actual tournament, guided in part by scientific objectives and in part by a desire to give all entrants an equal opportunity to do well in the tournament. However information on I and E(i) will

not be released to entrants before the tournament, to discourage them from "tuning" their programs to specific environments.

20. For each environment E(i), a total of N(i) DA games will be played, distributing a share A(i) of the $10,000 prize money with

    $A(1)+ \ldots +A(I) = \$10000.$

    The number of games N(i) will be chosen sufficiently large to average out variations in profits due to random choices of player matchings, token values, and tie-breaking rules.

21. In each of the N(i) DA games in environment E(i), entrants' programs will be randomly chosen- without replacement- from the pool of valid players for inclusion in the current DA game. If a player program is drawn that cannot fill the position (e.g., if the entrant only wrote a "buyer" program but was selected to play the role of "seller") the the program is withdrawn from the pool and another player is picked. Players which have already been selected are also withdrawn from the pool.

    Draws will continue in this way until the requisite number of buyer and seller programs are selected to play in the current DA game. This implies in particular that no program will play more than once in a single DA game.

22. If TP(i,j) is the total token profit earned by a player j in the subset of the N(i) games in environment E(i) in which it participated, then the actual dollar payment for that environment, DP(i,j), is given by:

    $DP(i,j) = c(i)TP(i,j)$

    where c(i) is the *conversion ratio* between token profits and dollar profits. The total dollar payment to entrant j in the entire tournament is then simply the sum over each of the I environments, $DP(1,j) + \ldots + DP(I,j)$.

23. The conversion ratio c(i) between token profits and actual dollar payments will be set in each environment E(i) to satisfy the following equation:

    $A(i) = c(i)TS(i)$

    Here TS(i) is the total surplus of the N(i) games in environment E(i). In a single game the surplus is geometrically the area between the implied supply and demand curves, to the left of the point where they intersect, and would be the total profit if all trading occurred at the theoretical competitive equilibrium price. TS(i) will be calculated by summing this surplus over the N(i) games in environment E(i).

    Since the conversion ratios c(i) will thus be determined a priori, the total dollar payment to participants may not be exactly $10,000, but is expected to be within a few percent of that figure.

24. Trading programs have the right of refusal. Thus, entrants do not have to write strategies to play in all possible environments. For example an entrant may submit a program that only plays the role of buyer. Of course, since the expected profit in any given game is positive, an entrant's tournament earnings will be reduced in proportion to the fraction of games in which their program is unable to play.

25. Token values are represented by T(j,k,l), where j indexes the player, k indexes the token number, and l indexes whether the player is a buyer or a seller. Tokens will normally be randomly generated according to

    $T(j,k,l) = A + B(l) + C(k,l) + D(j,k,l)$

    where

    $A = U[0,RAN1]$,
    $B(l) = U[0,RAN2]$ if l = buyer, 0 Otherwise,
    $C(k,l) = U[0,RAN3]$,
    $D(j,k,l) = U[0,RAN4]$.

    Here U[0,R] denotes a uniform random variable on the interval [0,R]. Random variables A, B(l), C(k,l), and D(j,k,l) are drawn independently of each other and independently for distinct indices (j,k,l).

    RAN1, ..., RAN4 are passed to the trading strategy using the 'gametype' variable; see definition of 'gametype' in section 3.2 "Public Constants" of chapter 3. Note that due to technicalities of the software design, no token value is allowed to exceed 8000. This implies that for very large settings for RAN(i), i=1,...,4 the actual distribution is truncated. In the tournament we will not choose any values for RAN(i) i=1,...,4 that yield truncated distributions.

    In a few environments, representing not more than 33% of the total available profit, the tokens may be generated by another method not revealed to the players. This will be indicated by gametype = 0.

26. The organizers expect entrants to obey the spirit as well as the letter of the above rules to produce strategies that exhibit "reasonable trading behavior." In the event that certain strategies are found to behave in a grossly unreasonable way (e.g., purely random strategies, or ones designed to maximize losses) or in a way that has not been explicitly covered in the above rules, the organizers reserve the right to exclude the deviant strategies after giving the entrant an opportunity to revise the strategy to produce more reasonable behavior.

7.2 Registration Form

Submit a printed copy of the following form by mail. A copy sent by electronic mail or by fax may be used to establish a tournament entry, but a printed copy with signature(s) and entry fee must follow promptly.

If your strategy has been developed by more than one person, submit only one trading strategy and entry fee, but include separate entry forms for each co-author.

Send forms to:

DAT
Santa Fe Institute
1120 Canyon Road
Santa Fe, New Mexico 87501
Phone: (505) 984-8800
Fax: (505) 982-0565
E-mail: **dat@sfi.santafe.edu**

Santa Fe Institute's

DOUBLE AUCTION TOURNAMENT

REGISTRATION FORM

I hereby register my entry for the Double Auction Tournament to be held at the Santa Fe Institute in March 1990. My entry consists of

1) This form, completed and signed.

2) The entry fee of $10.00 (check or money order in U.S. currency only). Make checks payable to "Santa Fe Institute."

3) Source code for a computer trading strategy according to the tournament rules. Check one:

__ 5.25-inch DOS disk enclosed (plain ascii file(s))
__ 3.5-inch Macintosh disk enclosed (TEXT format)
__ Sent by electronic mail to "dat@sfi.santafe.edu"
__ Submitted by other means: _____

4) Other: (check those that apply—all are optional)

__ written documentation
__ special instructions for compiling/linking/running
__ auxiliary files needed by program

I understand that this tournament is part of a research project to gather scientific data to improve our understanding of the market mechanism. I have read and accept the tournament rules that specify the form of valid computer program entries, and describe how the approximately $10,000 prize money will be distributed to tournament participants.

By submitting my trading strategy, I am giving researchers at the Santa Fe Institute the right to use it for scientific purposes in applications and experiments of their choice after the Double Auction tournament is held in March 1990. In particular, I grant them the right to describe my strategy in published reports, although I may choose not to be identified as the author of my trading strategy.

Signed _____ Date _____

Name _____

Affiliation _____

Address _____

_____

Phone _____ E-Mail _____

Do you wish to be identified as author of your strategy in published reports (and public list of entrants)? __ yes __ no

Did you use SFTE to develope or refine your strategy?__ yes __ no

If yes, did SFTE help you improve your strategy? __ yes __ no

CHAPTER 8: THE MESSAGE-PASSING PROTOCOL

This chapter gives technical details of the message-passing protocol which is used for communication between the player programs and the monitor program. It is not needed by participants who base their programs on a skeleton program.

8.1 Messages and message packets

The tournament involves a central monitor program and a number of competing player programs. The player programs communicate only with the monitor, not directly with each other. All communication is performed by sending and receiving simple messages.

A message consist of a single line containing a sequence of integers. Each integer is right justified in a 5-column field, blank filled on the left. A typical message might be

...14..350...-1

where each . represents a space. All integers lie in the range -999 to 9999 to leave at least one blank between columns.

The first integer in each message indicates the message type. Here we name such message types according to the code table in Section 8.6. Thus a typical message is written:

OFFER offer-price offerer

The capitalized name OFFER stands for the unique integer code for this message type (16 from Section 8.6), whereas the lower-case names 'offer-price' and 'offerer' stand for parameter values that are conveyed by the message.

Each communication from a player program to the monitor consists of a single message with one parameter (i.e. two integers in all). In some cases the parameter is unused and is set to 0.

Most communications from the monitor to a player program consist of several messages called a message packet. The individual messages all have 2 parameters (i.e. 3 integers in all). In some cases the second parameter is unused and is set to 0.

In the networked version of the game, in which the monitor and player programs may be on different machines, there are some initial pre-game messages exchanged before the main game begins. These pre-game messages are not purely numeric and do not obey the above conventions. They are described in Section 8.4.

8.2 Order of message packets

The following table shows the sequence of messages in a game:

Step	Monitor packets	Trader messages
Initialization	Initialization-1 Initialization-2	ACCEPT or REFUSE READY
Round	Round	READY
Period	Period	READY
Bid-offer	Bid-offer Bid-offer result	BID or OFFER or NONE
Buy-sell	Buy-sell Buy-sell result	BUY or SELL or NONE
End	End-of-game	

)))
)))
)))
))) k
)) j)
)))
) i))
)))
)))

i, j, and k show the number of times each group is repeated:
i = ntimes
j = nperiods
k = nrounds

The center and right columns show the message packets and messages in each step, reading down the page in order. The game may be divided into 6 types of steps, as shown in the left column. Each game has one initialization step and one end step. Between these there are 'nrounds' rounds, each consisting of one round step and then 'nperiods' periods. Each period consists of one period step and then 'ntimes' alternating bid-offer and buy-sell steps. These steps are described in detail in the remaining sections of this document. Appendix B provides a summary of all steps and their messages.

The center column shows the message packets sent by the monitor to the player programs. These may be either single or multiple messages, as detailed below.

The right-hand column shows the messages sent by the player programs to the monitor. The messages shown are alternatives. In a bid-offer step, for example, each player program is expected to send either a BID message or an OFFER message or a NONE message.

8.3 Individual Steps of a DA game

8.3.1 Initialization step

The initialization step occurs once at the start of a game. It consists of:

1. An initialization-1 packet from the monitor that provides the general parameters for the game;

2. A response from the player program that either agrees to play (ACCEPT) or drops out at this stage (REJECT);

3. An initialization-2 packet from the monitor that provides more specific information about the game and gives parameters particular to each player;

4. A READY response from the player program when it is ready to play.

INITIALIZATION-1 PACKET

The first initialization packet gives general information about the game. The following messages will be sent in order:

TYPE	protocol	monitor
GAME	game-type	game-id
LENGTH	nrounds	0
LENGTH	nperiods	ntimes
TOKENS	max-tokens	0
NUMBER	max-buyers	max-sellers
ROLE	role	timeout

The variables have the following meanings:

protocol — Specifies the version of the message-passing protocol being followed. This should match that of the player programs. protocol = 5 for the version described in this document.

monitor — Specifies the version of the monitor in use. This can be ignored by participants.

game-type — Used to identify different types of games that will be played, conveying the values of the token-generation random number parameters. See the gametype variable in chapter 3 of the documentation for details.

game-id — Provides a unique identification for this particular game. game-id will be increased for each game played.

nrounds — Specifies the maximum number of rounds in the game. Maximum 20.

nperiods — Specifies the maximum number of periods in each round. Maximum 5.

ntimes — Specifies the maximum number of time steps in each period. In each time step there is first a bid-offer step and then a buy-sell step. A period will never be terminated early (before ntimes time steps have been played) unless no bids, offers, buys, or sells are occurring, and no mutually profitable trades could be made. Termination will no necessarily occur in such a case. Maximum 400.

max-tokens Specifies the maximum number of tokens that any player will be allotted in any round of this game. Maximum 8. A maximum of 4 will be used in the tournament.

max-buyers Specifies the maximum number of buyers in this game; the actual number (given in the initialization-2 packet) may be less because some players may drop out after this step. Maximum 20.

max-sellers Specifies the maximum number of sellers in this game; the actual number (given in the initialization-2 packet) may be less because some players may drop out after this step. Maximum 20.

role Takes one of two values, and should match the type of player program it is sent to:

1 For a buyer.
2 For a seller.

timeout Specifies the conditions under which the monitor will timeout a player in any step:

0 Timeout occurs after a short time, which may vary from step to step. Governed by the timefactors in the monitor's game file.

9999 No timeout because the player was given the -w flag in the monitor's player file. The monitor will wait indefinitely for the player. other Approximate number of seconds until timeout. This is the timeout value from the monitor's game file. When a player is timed out in a bid-offer or buy-sell step it is given a "late" (-2) status when it next does respond. The player is terminated if it times out in any other step, or is not responding at the end of a period.

FIRST RESPONSE MESSAGE

The player program must respond to the initialization-1 packet with one of the following messages:

ACCEPT player-number
or REFUSE reason

In the first case the player program agrees to play the game, and should not drop out at any future stage. In the second case the program refuses to play and gives a reason; it should then exit immediately.

player-number Identifies the particular player program. Should be unique for each player. Use the value assigned to you, or 0 if a correct value is unknown. 9999 is normally used for a human player.

reason A reason for refusing to play. Possible values are:

1 protocol is unacceptable.
2 game-type is unacceptable.
4 nrounds is unacceptable.
8 nperiods is unacceptable.
16 ntimes is unacceptable.
32 max-tokens is unacceptable.
64 max-buyers or max-sellers is unacceptable.
128 role is unacceptable.
256 timeout is unacceptable.
<0 Other program error or problem.

Several of the values 1--256 may be added together if multiple conditions apply.

INITIALIZATION-2 PACKET

The second initialization packet gives specific information about the game and the player. The following messages will be sent in order:

NUMBER	nbuyers	nsellers
BUYERS	number-1	number-2
BUYERS	number-3	number-4
. . .		
SELLERS	number-1	number-2
SELLERS	number-3	number-4
. . .		
LIMITS	minprice	maxprice
PLAYER	id	0

The variables have the following meanings:

nbuyers — The actual number of buyers.

nsellers — The actual number of sellers.

number-n — The player-number of buyer or seller n. Enough BUYERS messages are supplied to give the player-number of each buyer. Enough SELLERS messages are supplied to give the player-numbe of each seller. If the number of buyers or sellers is odd, the last number-n parameter will be 0. Any or all of the other number-n parameters may be zero if the appropriate value is unknown or the players are anonymous.

minprice — The minimum price value allowed. Minimum 1.

maxprice — The maximum price value allowed. Maximum 9999.

id — Your own identification number, between 1 and nbuyers if you are a buyer, and between 1 and nsellers if you are a seller. Players are referred to during the game by these identification numbers, not by their player-number.

SECOND RESPONSE MESSAGE

The second response message just consists of an acknowledgement when ready:

READY id

id must be your correct identification number.

Start of round step

A start of round step occurs at the start of each round of the game. It consists of:

1. A round packet from the monitor, giving the number of the current round, and token values for the round;

2. A READY response from the player program when it is ready to continue play.

ROUND PACKET FROM MONITOR

The round packet consists of the following messages:

```
ROUND     r         ntokens
PRICES    price1    price2
PRICES    price3    price4
. . .
```

The variables have the following meanings:

r — The number of this round: r = 1, 2, ..., nrounds.

ntokens — The number of tokens you are given for this round.

price1—4 — Your redemption values, in decreasing order for a buyer, and in increasing order for a seller. There will be ntokens such values in all (shown as 4 here), given in as many PRICES messages as needed. The last parameter of the last PRICES message will be 0 if ntokens is odd.

RESPONSE

The response message just consists of an acknowledgement when ready:

```
READY     id
```

id must be your correct identification number.

Start of period step

A start of period step occurs at the start of each period of each round of the game. It consists of:

1. A period packet from the monitor, giving the number of the current period;

2. A READY response from the player program when it is ready to continue play.

PERIOD PACKET FROM MONITOR

The period packet consists of a single message:

```
PERIOD    r    p
```

The variables have the following meanings:

r — The number of the current round.

p — The number of this period: p = 1, 2, ..., nperiods.

RESPONSE

The response message just consists of an acknowledgement when ready:

READY id

id must be your correct identification number.

Bid-offer step

A bid-offer step occurs at each time in each period. It consists of three stages:

1. A bid-offer packet from the monitor defining the start of the step and giving the current time;

2. After processing, a response from each player that specifies a bid, an offer, or neither;

3. A bid-offer result packet from the monitor that reports on what happened during this step, and gives the current status.

BID-OFFER PACKET

A bid-offer packet consists of a single BIDOFF message:

BIDOFF t nobidoff

The variables have the following meanings:

t The current time: t = 1, 2, ..., ntimes.

nobidoff A variable saying whether a bid or offer is allowed by this player in this step:

0 Bid or offer allowed.
1 No bid or offer allowed---no tokens left.

RESPONSE

The response from the player must be one of the following messages:

BID bid-price
or OFFER offer-price
or NONE 0

A buyer may send a BID message to make a bid or a NONE message to do nothing. A seller may send an OFFER message to make an offer or a NONE message to do nothing. The message must be NONE if 'nobidoff' was non-zero in the BIDOFF message.

The variables have the following meanings:

bid-price The price of the bid being made. This must obey minprice <= bid-price <= maxprice and must be higher than the current bid price if there is one.

offer-price The price of the offer being made. This must obey minprice <= offer-price <= maxprice and must be lower than the current offer price if there is one.

BID-OFFER RESULT PACKET

The bid-offer result packet from the monitor reports on the disposition of this player's response (BID or OFFER or NONE), and on the other bids and offers made in this step. It consists of a BODISP message, possibly some BID and/or OFFER messages, and then a CBID and a COFFER message:

BODISP	status	trades
BID	bid-price	bidder
BID	bid-price	bidder
. . .		
OFFER	offer-price	offerer
OFFER	offer-price	offerer
. . .		
CBID	current-bid	bidder
COFFER	current-offer	
	cofferer	

There is one BID message for each new bid made, and one OFFER message for each new offer made. Two of each are shown above, but there could be from none up to 'nbuyers' or 'nsellers' such messages. The BODISP, CBID, and COFFER messages always appear.

The variables have the following meanings:

status — The disposition of the previous BID, OFFER, or NONE response. Possible values are:

- 0 NONE message received OK. You do not have an outstanding bid or offer.
- 1 NONE message received OK. Your previous bid or offer is still the current bid or offer.
- 2 BID or OFFER message received OK. Your bid or offer is now the current bid or offer.
- 3 BID or OFFER message received OK. Your bid or offer was bettered by another player.
- 4 BID or OFFER message received OK. Your bid or offer was equaled by another player, and you lost the random tie-break.
- -1 BID or OFFER message received, but unacceptable because nobidoff was non-zero or because the bid-price or offer-price was invalid; see the requirements listed in the response subsection above.
- -2 BID or OFFER message received, but too late. Your request was ignored and you missed one or more steps.

trades — Number of trades made so far by this player.

bid-price — The price of a bid that was made. The BID messages are arranged in increasing order of bid-price, with the last one (if any) producing the current bid price.

bidder — The id number of the buyer making the bid.

offer-price — The price of an offer that was made. The OFFER messages are arranged in decreasing order of offer-price, with the last one (if any) producing the current offer price.

offerer — The id number of the seller making the offer.

current-bid — The current bid price, or zero if none. If there were any bids this is equal to the highest bid-price. If there were no new bids the previous value of current-bid (which may be 0) is retained.

cbidder — The id number of the buyer whose bid is the current bid, or 0 if none.

current-offer — The current offer price, or zero if none. If there were any offers this is equal to the lowest offer-price. If there were no new offers the previous value of current-offer (which may be 0) is retained.

cofferer — The id number of the seller whose offer is the current offer, or 0 if none.

8.3.2 Buy-sell step

A buy-sell step occurs at each time in each period, after the corresponding bid-offer step. It consists of three stages:

1. A buy-sell packet from the monitor defining the start of the step and giving the current time;
2. After processing, a response from each player specifying a buy request, a sell request, or neither;
3. A buy-sell result packet from the monitor that reports on what happened during this step, and gives the current status.

BUY-SELL PACKET

A buy-sell packet consists of a single BUYSELL message:

BUYSELL t nobuysell

The variables have the following meanings:

t — The current time: t = 1, 2, ..., ntimes.

nobuysell — A variable saying whether a buy or sell request is allowed by this player in this step:

- 0 Buy or sell request allowed.
- 1 No buy or sell request allowed—no tokens left.
- 2 No buy or sell request allowed—no current offer or bid.
- 4 No buy or sell request allowed—not the holder of the current bid or offer. If several of the codes 1, 2, 4 apply they will be added together.

RESPONSE

The response from the player must be one of the following messages:

	BUY	current-offer
or	SELL	current-bid
or	NONE	0

A buyer may send a BUY message to request a buy, or a NONE message to do nothing. A seller may send a SELL message to request a sell, or a NONE message to do nothing. The message must be NONE if nobuysell was non-zero in the BUYSELL message.

The variables have the following meanings:

current-offer — The current offer price, as given in the preceding COFFER message. This must be given correctly.

current-bid — The current bid price, as given in the preceding CBID message. This must be given correctly.

BUY-SELL RESULT PACKET

The buy-sell result packet from the monitor reports on the disposition of this player's response (BUY or SELL or NONE), and on the BUY or SELL request that was accepted, if any. Other players' requests that were not accepted are not reported. It consists of a BSDISP message, possibly a TRADE and a PLAYERS message, and then a CBID and a COFFER message:

BSDISP	status	trades
TRADE	type	price
TRADERS	buyer	seller
CBID	current-bid	cbidder
COFFER	current-offer	cofferer

Both the TRADE and TRADER messages appear if a transaction occurred; otherwise neither appears. The BSDISP, CBID, and COFFER messages always appear.

The variables have the following meanings:

status — The disposition of the previous BUY, SELL, or NONE response. Possible values are:

0 NONE message received OK.

1 BUY or SELL message received OK and accepted. You just made a trade.

2 BUY or SELL message received OK, but not accepted because both the current bidder and the current offerer requested a buy or sell and you lost the random 50--50 toss.

3 BUY or SELL message received OK, but not accepted because another buyer or seller (the same type as you) also made a valid request, and won the random tie-break.

-1 BUY or SELL message received, but unacceptable because nobuysell was non-zero, or because the current-bid or current-offer specified did not match the actual current bid or offer price.

-2 BUY or SELL message received, but too late. Your request was ignored and you missed one or more steps. Note that you may have made a trade even if 'status' is not 1, because another player may have accepted your current bid or offer. Check whether 'buyer' or 'seller' (as appropriate) is equal to your 'id' to know for sure.

trades — Number of trades made so far by this player.

type — A code specifying which type of transaction occurred:

1 A BUY request was accepted.

2 A SELL request was accepted.

price — The price of the transaction that occurred. This will be equal to the previous current offer price if 'type' = 1, or to the previous current bid price if 'type' = 2.

buyer — The identification number of the buyer involved in the transaction.

seller — The identification number of the seller involved in the transaction.

current-bid — The current bid price, or 0 if none. This will definitely be 0 if there was a TRADE message. Otherwise it will be non-zero if it was previously non-zero.

cbidder — The id number of the buyer whose bid is the current bid, or 0 if none.

current-offer — The current offer price, or 0 if none. This will definitely be 0 if there was a TRADE message. Otherwise it will be non-zero if it was previously non-zero.

cofferer — The id number of the seller whose offer is the current offer, or 0 if none.

8.3.3 End of game step

The end of game step consists of a single message from the monitor to each player:

END profit efficiency

The variables have the following meanings:

profit — The total profit made by the player in this game. Values larger than 9999 or smaller than -999 are reported as 9999 and -999, respectively.

efficiency — The ratio of 'profit' to the profit predicted by economic theory, times 100. This is a measure of how well the player performed, with 100 being "average" and higher values being better.

The player program should terminate itself upon receiving this message.

8.4 Special Messages

8.4.1 QUIT messages

At any stage where an ordinary message is expected, the player program may send the monitor the message:

QUIT type

The monitor will then drop that player from the game. 'type' should be zero for an intentional abandonment of the game (normally used only for a human player), or non-zero if an unexpected fatal error occurred in the player program. The monitor will report "quit" for the first case, and "fatal error" for the second. The player program itself should print an appropriate error message (to stderr if UNIX).

8.4.2 KILLED messages

The monitor (or DANI) may at any time send a player the message:

KILLED reason 0

The player program should print an error message (to stderr if UNIX) if desired and then exit immediately. The reason parameter is a code saying why the monitor killed the player:

1. Killed by the operator running the monitor.
2. The player was not responding at the end of a period. A temporary lack of response is allowed within a period (and gives status = -2), but is fatal at the end of a period.
3. The player sent a message with a message type that was either illegal in the current context, or entirely unknown (not in Appendix C).
4. The player sent an undecodable message (not 2 integers).
5. The monitor couldn't find or couldn't read the .out message file which should have been written by the player (file-based communication only).
6. The monitor couldn't read a message from the player because of an I/O failure (pipe-based or inet-based communication only).
7. The monitor encountered an error, probably because the game wasn't properly defined.
8. The game was abandoned, probably because there weren't enough players.
9. Killed by DANI due to a communications problem or inconsistency.

>10. Another reason—reserved for possible future use.

0. No error—silent kill; used by DANI if there isn't actually a game in progress.

8.4.3 TEST messages

TEST messages are used by the monitor or by DANI to check that a player is functioning properly. At any time the monitor or dani may send the message:

TEST k 0

and the player should respond with:

TEST k

The variable k is an arbitrary integer, which must be echoed correctly in the player's reply.

8.4.4 Pre-game messages

In the networked version of the game, in which the monitor and player programs may be on different machines, there are some initial pre-game messages exchanged before the main game begins. They apply equally whether the player is actually an interface to a human player, or is a machine player. They are as follows:

1. FROM PLAYER TO MONITOR

The player program must initiate the dialogue with:

DA role type userid name

Note that 'DA' means the literal upper-case characters DA, not a numeric code.

role — Specifies the roles that the player is willing to play: 1 for a buyer, 2 for a seller, 3 for either, 0 for inquiry only.

type — Specifies the type of player: 1 for a human player directly connected, 2 for a machine player directly connected, 3 for any player connected via DANI.

userid — The program's userid on the player's local machine. Up to 8 characters.

name — The player's or player program's name, up to 30 characters before the newline.

2. FROM MONITOR TO PLAYER

The monitor responds with one or more textual messages, which should normally be printed or displayed by the player, followed by one of the following: either by

start
nogame
abort

'start' signifies that the main game (starting with the initialization-1 packet) is about to begin. 'nogame' indicates that there is no current game in which to participate. 'abort' signifies that, although a game was expected to begin, either the whole game has been aborted or the player is not eligible to play in it. The reason for an 'abort' message will be given in the preceding textual messages. The player program should exit after receiving a 'nogame' or 'abort' message.

8.5 Summary

This summarizes the messages passed in a game, in order. Indented messages are from the player programs to the monitor.

TYPE	protocol	monitor
GAME	game-type	game-id
LENGTH	nrounds	0
LENGTH	nperiods	ntimes
TOKENS	max-tokens	0
NUMBER	max-buyers	max-sellers
ROLE	role	timeout

	ACCEPT	player-number		
or	REFUSE	reason		
	NUMBER	nbuyers	nsellers	
	BUYERS	number-1	number-2	(Note 1)
	SELLERS	number-1	number-2	(Note 1)
	LIMITS	minprice	maxprice	
	PLAYER	id	0	
	READY	id		
A:	ROUND	r	ntokens	
	PRICES	price-1	price-2	(Note 2)
	READY	id		
B:	PERIOD	r	p	
	READY	id		
C:	BIDOFF	t	nobidoff	
	BID	bid-price		
or	OFFER	offer-price		
or	NONE	0		
	BODISP	status	trades	
	BID	bid-price	bidder	(Note 3)
	OFFER	offer-price	offerer	(Note 3)
	CBID	current-bid	cbidder	
	COFFER	current-offer	cofferer	
	BUYSELL	t	nobuysell	
	BUY	current-offer		
or	SELL	current-bid		
or	NONE	0		
	BSDISP	status	trades	
	TRADE	type	price	(Note 4)
	TRADERS	buyer	seller	(Note 4)
	CBID	current-bid	cbidder	
	COFFER	current-offer	cofferer	

Go to A, B, C, or D.

D: END profit efficiency

Notes:

1. Enough of these messages appear to give a number for each buyer and each seller.
2. Enough of these messages appear to give 'ntokens' price values.
3. Zero or more of each of these messages appear.
4. Neither or both of these messages appear.

8.6 Code Table

This table assigns numerical values to each of the keywords mentioned in the text. The alphabetical order used is purely arbitrary, and may not be maintained in the future if further keywords are added. The values listed here will not be changed.

ACCEPT	1
BID	2
BIDOFF	3
BODISP	4
BSDISP	5
BUY	6
BUYERS	9
BUYSELL	7
CBID	8
COFFER	9
END	10
GAME	11
KILLED	98
LENGTH	12
LIMITS	13
NONE	14
NUMBER	15
OFFER	16
PERIOD	17
PLAYER	18
PRICES	19
QUIT	99
READY	20
REFUSE	21
ROUND	27
ROLE	22
SELL	23
SELLERS	30
TEST	31
TOKENS	28
TRADE	24
TRADERS	25
TYPE	26

CHAPTER 9: REFERENCES

9.1 References

This list provides some bibliography for those interested in pursuing further the previous work on Double Auctions and related topics. See reference by Rust, Palmer and Miller (available as Santa Fe Institute working paper, hardcopy but no Email version) for a basic overview of Double Auctions, existing analysis of strategies in Double Auctions, and basic motivation for the Double Auction tournament.

Axelrod, R. (1984) *The Evolution of Cooperation* (Basic Books; New York).

Easley, D. and Ledyard, J. (1986) "Theories of Price Formation and Exchange in Double Oral Auctions" Social Science Working Paper 611, California Institute of Technology.

Dorsey, R.E. (1988) "Estimation and Simulation of the Double Auction" discussion paper for the ESA meeting, Department of Economics, University of Arizona.

Plott, C.R. (1982) "Industrial Organization Theory and Experimental Economics" *Journal of Economic Literature* 20, 1485-1527.

Plott, C.R. (1989) "An Updated Review of Industrial Organization Applications of Experimental Methods," California Institute of Technology.

Rust, J., Palmer, R.G., and Miller, J.H. (1989) "A Double Auction Market for Computerized Traders" Santa Fe Institute Working Paper.

Smith, V.L. (1962) "An Experimental Study of Competitive Market Behavior" Journal of Political Economy 70, 111-137.

Smith, V.L. (1976) "Bidding and Auctioning Institutions: Experimental Results" in *Bidding and Auctioning for Procurement and Allocation*, ed. Y. Amihud, 43-64 (New York, New York University Press)

Smith, V. L. (1982a) "Microeconomic Systems as an Experimental Science" *American Economic Review* 54, 923-955.

Smith, V.L. (1982b) "Competitive Market Institutions: Double Auctions vs. Sealed Bid-Offer Auctions" *American Economic Review* 72, 58-77.

Smith, V.L., Suchanek, G.L., and Williams, A.L. (1989) "Bubbles, Crashes and Endogenous Expectations in Experimental Spot Asset Markets" *Econometrica.*

Waldspurger, C.A., Hogg, T., Huberman, B.A., Kephart, J.O., and Stornetta, S. (1989) "SPAWN: A Distributed Computational Economy" Xerox System Science Laboratory preprint P89-00025, Palo Alto.

Wilson, R. (1984) "On Equilibria of Bid-Ask Markets" Technical Report 452, Stanford University.

Wilson, R. (1985) "Efficient Trading" in *Issues in Contemporary Economics and Welfare*, ed. G. Feiwell, 169-208 (State University of New York Press, Albany, New York).

Appendix
Recent Questions and Answers to DAT-LIST

I just played a game through the human interface, and there was one thing I didn't understand. the periods were supposed to be 50 "times" long, but in the second round, they were only 15 and 11 long. Why'd that happen?

The terminations in the DA games you played in are normal: it is due to the setting of the "deadsteps" parameter in the "game" file of the SFTE. The idea is that if the monitor detects that all players have already made all possible mutually profitable trades, it will terminate the game early. In the actual tournament we will not terminate early, the full number of bid/offer and buy/sell steps (given in the variable 'ntimes') will always be played out. In the SFTE there may be human players, and having early termination allows one to move onto to the next game without having to play out the remaining 'deadsteps' in the current game.

We are interested in participating in the DA tournament. A couple of quick questions? Are teams allowed to enter (as a single entity)? What are the rules of the market?

Yes, see rule 12 of chapter 7, Rules. Other market rules can be found in section 1.2 of chapter 1, Introduction.

I hope you have plans to make SFTE a little more realistic. A given strategy's success depends strongly on its competition and on the set of parameters of the game. The quality of the submissions will be much higher if the participants are able to get a better idea how a competitive DA environment works.

If our interest here is sustained, we can probably set up our own fairly competitive exchange on a local machine. However, for those places without the resources to set up a local exchange, the following would be helpful:

(1) Set up the SFTE so that it uses a randomly selected, varying set of parameters. If you leave it as-is, my guess is that you'll have a number of not-very-general submissions which are tuned to the current setup. That would be unfortunate.

(2) Encourage entrants to specify that their entry may be entered in a 'pre-tournament pool', from which programs are drawn for each SFTE run. This will allow the quality of entries to gradually increase. Research-minded entrants will be happy to allow this.

The preceding two points were intended to address the following maxim of contest-running and assignment-giving: "Participants must be given a very accurate idea of the environment in which their programs will be run, or else many submissions will be buggy and useless."

The following point is to further improve entry quality:

--> Add a facility to the SFTE so that entrants can send requests for game environments via Email, such as "Run a game using parameters X at hour Y today". If you don't wish to implement such an automatic Email-parsing setup, encourage some other site to do so. You could arrange to get transcripts of the results, if you prefer to keep close tabs on what goes on.

This is to help foster a spirit of experimentation. I think encouraging someone to do this (for instance, myself) would be a wise idea. I'm NOT encouraged, however, by the fact that currently my reward for this effort would be that you ask me for $200.

You're quite right that the SFTE is very unrealistic. In fact the setup that's presently running was never intended to remain, but we've all been too busy to change it.

Within a few days We'll set it up so it has different parameters at each time of day (or maybe a six-hour cycle). For most of those we'll take out the fixed token values and just use the random distributions, typical of what we'll use in the tournament (RAN1—RAN4).

We're not convinced that an automatic request system for game environments would be much used, and don't see exactly what you have in mind. How would you deal with conflicting requests? But if you want to develop the idea further we'd be pleased to consider it. Of course we wouldn't charge you $200 if you made a software contribution.

It is clearly sensible to let participants suggest environments for the SFTE (not necessarily automatically parsed)—we'd be happy to receive specific suggestions.

If you do want to think more about an automatic system, we think that all you need to know initially is this:

1. The game is governed by a 'game' file and a 'players' file, just as with a local monitor. The game file has some extra parameters on the end to specify the minimum and maximum number of players etc -- we can mail you a copy if you're interested. So an automatic system would merely have to generate those files. We can readily have different files for different times of day—the monitor is started up by 'cron' for each game. (A separate program runs when the monitor doesn't, to answer queries; this program is started up and killed by the monitor as needed.)

2. At present mail sent to 'dat' is first read by a secretary or staff member at SFI, who then takes appropriate action for routine requests or forwards it to one of the organizers. But we could have a separate mail address for environment requests, and easily arrange for that mail to be piped into a parsing program.

To my mind, the biggest problem with the SFTE is the local players. They're pretty dumb. There are three; two copies of the skeleton program that was distributed, and one of the simple heuristic (non-adaptive) strategy, that usually does worst of all. The problem is how to improve this. No-one seems to want to contribute better players; it' would obviously lead to other's adapting their programs to beat them. At present the organizers are too busy to work on it. Perhaps I'll send a plea to the mailing list after I announce the changes I'll soon make to the parameters. Any better suggestions (or players!!) would be appreciated.

[The following announcement followed pursuant to the above reply:

The Santa Fe Token Exchange has been improved to play a wider variety of games. At different hours of the day, currently on an 8-hour cycle, there will be different values of the game parameters, and different numbers of local players.

Most of the games will have tokens generated solely with the RAN1-RAN4 random number parameters, whose values will be communicated via the 'gametype' variable. This is how the majority of tournament games will be run.

A schedule of games is included in the message you get using 'inquire' or by trying to connect when there is no game pending.

The biggest remaining problem with the SFTE is that the local players are pretty dumb. Most of them are just the C skeleton program. We badly need some more challenging players on the SFTE; does anyone have any code to volunteer? The organizers plan to write some more, but it won't be soon. Everyone would appreciate contributions, and your code would get to play lots of games.

In general, more sample players with different strategic approaches would be appreciated by many participants, both on the SFTE and for local use or study. PLEASE CONTRIBUTE if you have something.]

What is the significance of either the 1 or 4 tokens you can either buy or sell in each period? why is there a limit, and why is it (seemingly arbitrarily) small?

The DA software allows tokens to be variable, with a maximum of 4. There is no particular reason for this limitation except to simplify the writing of strategy programs. Obviously having only 1 token is the simplest, and having more tokens is more complex. We decided to have a mix of games with 1 and 4 tokens. Another limitation is the requirement to bid/offer and buy/sell only 1 token each bid/offer and buy/sell step. We could have easily generalized the software to allow such "multiple unit double auctions" (MUDA's), but we decided again to start with the simplest possible framework so as not to discourage too many potential entrants.

There appears to be no way to capture a pure arbitrate situation. is this a feature? it is claimed in the instructions that the lock-step timing (which would seem to eliminate zero-time transactions) is necessary "to eliminate the effect of processor and network delays" but in the real situation, there are people who have a better position precisely because of this. bigger companies can spend larger amounts on research and computation power—this is a way of life (something that most think is keeping the little guy out of the market, but that's another posting), so why is it being limited by the tournament?

We don't know what you mean by "pure arbitrage": arbitrage as it is usually understood, say, in options trading is not a feature of this game. There is arbitrage, however, in the sense of selling or buying tokens at prices which are either above or below, respectively, the market price. The crucial difference is that this is not riskless arbitrage since no single trader actually knows a priori what the "market price" will be. As far as the issue of "lock step timing", this is not really a limitation since one can approximate a continuous-time trading environment arbitrarily well by our discrete-time formulation provided the number of time steps are made sufficiently large. Furthermore there is a sense in which some players may have a computational or communication advantage: the monitor software allows us to set response time limits after which responses are marked as 'late' and orders are not executed. In the INET version of the monitor there are variables 'mintime' and 'maxtime' specifying the minimum and maximum times to wait for players communicating over the Internet as well as 'timefactors' for piped-based players running locally at SFTE. Players who have better communication links to SFTE are more likely to get their response sent in on time, and thus have a trading advantage. Similarly, a player running on a Cray-2 can make far more calculations in the given response time than can a player running on Vax. The DA software can be configured to wait indefinitely for player responses, thus equalizing such computational or communication advantages. See the 'gamefile' for details on setting the 'timefactors' which govern responses. In the actual DA tournament we will have equal response time limits for each strategy. Some, however, will exceed the limit and have late responses. These inefficient strategies will be at a disadvantage, therefore. See the 'game' and chapter on the monitor for further details.

In the intro document, the word (or root) "strategy" appears 30 times, while nothing is mentioned of tactics. is there anyone who believes that success can be had by strategy alone? is it significant that the judging seems only concerned with "new strategies" or is this a linguistic oversight?

To us, 'strategy' means a response rule that it planned out before the fact, whereas 'tactics' denotes responses made 'on the fly'. Thus a human player is capable of tactical behavior, but by definition a computerized trader can only behave strategically. This distinction is somewhat clouded by learning/adaptive algorithms that self-update their behavioral rules on the basis of experience in the DA. We might call this 'tactical' behavior but we decided to call it 'strategic'. It's not a linguistic oversight, it's a basic philosophical issue as to what is the boundary between 'artificial intelligence' and 'human intelligence'.

Participating in SFTE from here may be difficult as most people at [company name] do not have direct Internet access (i.e. FTP), but can only send and receive e-mail through a gateway machine.

Your problem with no direct internet access is not unique; it seems to be a common setup at commercial sites. I've been discussing ways around it with some people at [company], involving running a relay on the gateway host. In their case the relay could be a modified version of DANI

Of course there's no real solution unless you can run programs (or at least a server) on the gateway machine. A relay server wouldn't use much in the way of resources, but might violate company policies.

Hi, I just read over the rules for the token exchange. One question: What does it mean that "if both sides accept, the tie is broken randomly"? I would expect that if buyer accepts offer and seller accepts bid, then transaction is completed; how is it a tie that needs breaking?

The question is the price of the transaction. If the buyer's request to accept the current offer is accepted, the transaction price is the current offer price. If the seller's request is accepted it's the current bid price.

The idea of tie breaking (as opposed to splitting the difference, for example) is that in the real world one request would have occurred first. It's only in the artificial case of discrete steps that they can occur simultaneously.

I would like to run a local tournament in [country] and submit the winner. Can we have the software "arena" that it will be run in?

I would like more information about getting the software necessary to set up an internet token exchange on our local (Sun) network. I am devising an adaptive strategy learning algorithm to play DA and it needs to see lots of players play lots of games. Having a TE locally would make training a lot easier. Thanks.

It may make sense to set up a local token exchange on your network. We've wanted to restrict people setting up other "public" exchanges, mainly so that activity is focused in one place. We also realize that the software can easily be used to implement other network games (or other distributed activities with a central manager), and have wanted to get some credit for it. For these reasons we're currently charging a fee of $200 for the networking stuff; that entitles you to set up ONE token exchange and have as many network players on your local network as you wish. It does NOT entitle you to distribute the networking software further, or to use it for any purpose other than for a Double Auction Token Exchange.

The networking software includes (if the necessary facilities are available on your system):

- Code that lets the monitor listen at an internet socket, waiting for players to connect.
- Code to control the number of network and non-network players, and to start the game when a criterion based on number of players and/or expired time is satisfied.
- Code to mail game output back to participants.
- Code to maintain a database of player usernames and electronic addresses, with optional redirection and wildcarding—used for the mailback.
- A separate program that runs automatically when no game is active, and answers enquiries with a current message.
- Code for the C-players, DANI, and INQUIRE, to let them connect to any Token Exchange, not just sfi.santafe.edu.

In addition we'll give you reasonable support and help in setting things up.

... I am interested in whether enough tournament rounds will be performed to adequately evaluate an adaptive player.

We haven't finally settled this yet, and it will have to depend to some extent on the number of entries. We're certainly interested in adaptive players—that's actually one of our main interests—so we very much want to have enough rounds to make that sensible. How many that is is not yet clear for this game. Besides the tournament itself, there's the possibility of additional games both before and after the tournament, either on the Santa Fe token exchange or on local monitors.

We're expecting to run the tournament on a batch of Sun 4's, but could switch to a Cray 2 if necessary.

So, in all, we definitely want to make adaptive approaches sensible, and will strive to make it so, limited only by resources.

... have you been approached on getting the local monitor software running on the Commodore Amiga personal computer? The Amiga offers a message-passing multitasking environment, and many UNIX utilities have been ported quite easily.

... I would like to see more sophisticated simulation and modeling tools available for my favorite personal computer, and the DAT simulator looks like a good candidate.

No, we haven't looked into getting things going on an Amiga. Unfortunately we don't have one available, or we might try.

[Some further discussion about the technical issues ensued. It looks quite possible, and will probably be developed by the respondent. The following remark is of general interest:]

One thing I would _not_ like to do is overwhelm the DAT/SFTE project with hordes of uninformed contest-players. Any work I or others here at [company name] might do would of course be released only under conditions compatible with your plans and wishes.

Your comment on releasing hordes of Amiga (and other) people on the DAT is important. We've worried a bit about that. We generally feel that it won't be a problem, particularly because a fair bit of effort is involved in understanding the problem and the software. But we might be wrong. We'll certainly limit tournament entries (to 100 max), but would be upset if we got 400 and had to accept the first (not necessarily the best) 100. On the other hand we may get only 10-20.

... I am surprised that in a contest calling for some sophisticated computation, you are limiting the choice of languages to procedural algebraic languages that are 20 years old, or more. If your goal is to obtain a readable, understandable description of a good strategy (the winning program), then the use of a modern language in which clear abstraction is possible would surely contribute to that goal. Perhaps if you ever repeat the contest, you might consider the following approach: define a communications protocol for the steps of the game which all programs must follow. Then any program, regardless of its source language, could play.

What you suggest in terms of defining a communications protocol for the steps was actually the first thing we did. It's all in the "messages" chapter of the documentation. It only requires that you read/write simple numerical messages to stdin/out.

Later we added skeleton programs in C, Fortran, and Pascal to make it easy for participants who wouldn't want to mess with all the communication and control issues—they just see a small set of subroutines that get called to make the strategic decisions.

We considered writing skeletons in several other languages too, including Lisp and Prolog, but were deterred for two reasons:

1. None of us is fluent in those languages, and
2. We don't presently have compilers for them on our Sun network at Santa Fe, except for the MIT C-scheme Lisp dialect.

(1) does not prevent anyone else (e.g. you) writing their programs in another language (and perhaps letting others use the resulting skeleton). The DANI program will then let you play games on the Santa Fe Token Exchange—you don't have to build internet TCP/IP stuff into your own program. You can also play directly with a local monitor.

(2) is only needed for the tournament itself. We can purchase additional compilers if they're (a) available, (b) not too expensive, and (c) in demand by [potential] participants. If (a) or (b) is a problem we may consider running some players in the tournament on remote machines, though we'd rather avoid that because it makes the tournament harder to automate, and raises security issues.

Before I go about reinventing some wheels, I wanted to check with you to see if anyone else has already built skeletons or monitors in Lisp. Has anyone? If not, we'll begin one, and most likely make the skeleton available to others.

Those of us working on this here would prefer a Lisp platform for the tournament. Based on some comments I've read, I'm hoping that this will be possible. For example, I'm sure that for a project like this, some Lisp vendor would be able to provide a evaluation copy (pun intended) for some defined length of time (1 or 2 months) (I've just done this with 2 different vendors). There are many other ways of getting a Lisp. I'd just like to verify that using a Lisp implementation in the tournament is still considered feasible. If you need any assistance in making a Lisp environment available, let me know and we'll be glad to help out.

To our knowledge no-one has yet developed a skeleton trading program in LISP. We have not done so because of the unavailability of a LISP compiler for the SUN workstations at the Santa Fe Institute. We do have an MIT C-scheme LISP dialect, but not the standard LISP compiler which costs $3000. We will allow LISP if we are able to acquire a LISP compiler/interpreter. Does anyone have any suggestions? We would like to get one for free since all of our money is going into prizes, but if there is overwhelming demand for LISP, we will attempt to raise money to purchase one.

If you do succeed in developing a LISP skeleton, we would be happy to distribute it along with the Pascal, Fortran and C skeletons.

Is there a Mac version available?

A version for the Macintosh is now completely integrated into the software. It runs ONLY under Multifinder, and only on a Mac-Plus or later model. It's OK with only 1MB of memory. Skeleton programs are available for Lightspeed C (version 3 or 4), Absoft MacFortran, and Lightspeed Pascal. Binary executables are available for the monitor (no graphics), the human player interface, and pp. No networking is yet supported; like the IBM PC version this is strictly for a single machine.

Besides the usual source files, Macintosh users should note that there's an extra source file for the monitor (mac.c), and a binary file (mac.sit) containing executables in a Stuffit archive, both available by ftp. There are also README.mac files in some of the directories. A Macintosh disk containing all the Macintosh executables, source, and documentation is available from Santa Fe (same fee as for PC disks 1 & 2).

Non-Macintosh users do not need to get the new versions of the software (besides tokens.c and rpe.c for the change discussed above). They should however be careful not to mix old and new versions of the files for the C player; get them all if you need any.

[Comment about automatic mail-back of SFTE game results to participants]

If there is any easy way to do it, you might consider telling the monitor that [foo.yyy.edu] should always be mapped to [xxx.yyy.edu] for any foo. This is because [yyy] has around 900 workstations, none of which run sendmail. [xxx.yyy.edu] is the central mail hub.

[Done. Comment added:]

There's a database that performs such mappings, and we can easily set it up to do any of the following mappings:

1. user1@address1 --> user2@address2 (with user2 equal or unequal to user1).
2. <any_user>@address1 --> <any_user>@address2.
3. <any_user>@addr* --> <any_user>@address2.

Here the addresses are internet addresses, specified by an ascii string like 128.109.182.2. "addr*" means one like 128.109.18*, which matches anything that's the same up to the *.

Is there any (basic) literature available? Could you provide references?

There are some references in the "refs" chapter of the documentation. We'll soon try to add some more at the basic level, and identify which are introductory.

I notice that the number of entries will be limited to 100.

How are you going to choose among the entries, in case you receive more than 100?

See chapter 7, rules 1 and 2.

How many periods will there be in a round? Enough to build charts of price action?

The number of periods per round, nperiods, is an environmental variable that may change from game to game, see rule 17. The maximum number of periods will never exceed 5, however. See definition of nperiods in section 3.2.

Although nperiods will vary from 1 to 5 in the tournament games, it will probably usually be 1 or 4. We'll have a variety of game types with some variation in this and other parameters. We think it should indeed be worthwhile to build charts of price action.

Suppose there are 100 players. Each plays the same number of games. What is the theoretical maximum prize money a player may win?

To obtain the maximum theoretical profit when you know everyone's token costs/redemption values, order the buyer's redemption values in decreasing order and the seller's token costs in increasing order (these lists are then called the demand and supply curves, respectively). Now pair the first on each list, and take the difference. Then the 2nd from each list, then the 3rd, etc., until the lists cross. The sum of the differences up to the crossing point is the theoretical maximum profit. The sum of all the buyer's profits and all the seller's profits can't exceed this.

Since money is distributed among all players, I guess it is very difficult to win more than $500. Am I right?

It depends on the number of entrants. Ex ante with N equally matched entrants, we would expect that each would earn approximately 10,000/N. The actual amount will be more or less depending on the quality of the strategy and the number of environments it can play in. A strategy that only plays the role of buyer will only participate in roughly half of the games it is chosen for, and therefore its potential earnings will be cut in half. The DA tournament will consist of a large number of DA games to average out variations in profits due to chance occurrences, such as the values of tokens drawn by a random number generator. Since we limit the maximum number of entries to 100, *ex ante* expected earnings should be now less than $100.

I still don't understand how prize money will be distributed? If money is paid in proportion to the total profit earned by each player, why the total money paid can't be exactly $10,000?

See rules 20 through 23 in chapter 7, rules. This should explain why total prize money paid out will not exactly equal $10,000. We will, however, guarantee that actual payments will be within a few percent of $10,000.

We could of course adjust the conversion between token values and dollars to make the total exactly $10,000. But that would involve seeing how much profit was actually made in each game, rather than using the theoretical maximum (times a number a little less than one). Using the actual profit would make it a sensible strategy to try to sabotage other players, or the game as a whole, so as to improve the conversion ratio (this is a standard problem in "zero-sum" games). We don't want that sort of destructive approach, so fix the ratio on the basis of theoretical profit. We're confident we can make the total within a few percent of $10,000.

I have a question: If I am the current offerer, can I refuse the buy request from other players? (Or: If I am the current bidder, can I refuse the sell request from other players?)

No you can't refuse buy/sell requests. Once you make a bid or an offer, that's a commitment to make a trade at that price if someone wants to accept it, until that bid/offer is beaten by a better one or a trade occurs. This is known as "New York Rules" because it corresponds to the way stocks are traded on the New York Stock Exchange.

Would it be possible to use QuickBasic on a PC?

Yes and no. It would be relatively easy to make a QuickBasic skeleton which would run on a PC. But that wouldn't help when we come to run the tournament, which has to run in a Unix environment. As far as we know there's no Basic (let alone QuickBasic) compiler for a Unix system. There may be Basic-to-C translators that could help.

[Does anyone know any more about Basic compilers/translators? Would anyone else like a QuickBasic skeleton?]

As an interested party (and possible player) of the DAT, I was wondering if the folks at Santa Fe are planning on making any distinctions between types of players based on the level of pre-programmed strategy.

Some computer players may be very sophisticated and operate with a variety of programmer selected strategies. Other players may be intentionally designed with initially simple strategies with the hope that learning algorithms will develop strategies on there own. The former may be likely to defeat the later soundly (maybe not?), but it seems to me the interest of the programmers who design the players are different, so at least some designation of the different approaches may be useful from an experimental point of view.

Maybe the following designations would be useful:

1) *Fixed strategy player.*
2) *Adaptive player (varying parameters of a mostly fixed strategy).*
3) *Learning player (initialized with very little strategy, with learning algorithms attached).*

Although the interactions of all three sorts of players in the same auction are interesting, tournaments consisting of players of each type may be exciting as well.

I would like to hear thoughts on these issues from other potential players and the auction organizers.

For those who decide to write the learning type algorithms, I think an additional test would be to see how their algorithms did as they progress through the tournament. Obviously the declaration of some sort of winner in the santafe tourney is dependant on consistent play from the first rounds, but I feel quite alot about particular adaptive, or learning algorithms might be learned by observing

their performance over a period of time against fixed algorithms. Questions that pop to mind at this time are along the lines of:

What is the slope of the learning curve?

Is there a plateau, or worse yet, a period of negative slope?

I haven't spend much time working out the details of how to analyze such a situation, but I agree that the concepts are interesting.

We recognize that there will be a wide variety of strategies playing in the DAT. It's difficult in the first place to define what "adaptive" strategies are and how they differ from other types of "learning algorithms". We feel that random matchings are appropriate for the tournament to give all entrants an equal chance. However in our private scientific investigations after the tournament we will be able to conduct precisely the kinds of experiments you suggest.

In advance of the tournament we can also have "trial heats" for certain classes of games and player types. We will be announcing such "trial heats" shortly. We invite you (or anyone else on dat-list) to suggest interesting scenarios for the trial heats, at pre-announced times on the SFTE.

Please, can you give me some information on how TS(i)—i.e., the total surplus of the games in environment i—is computed? [The explication in rule 23 is too terse.]

According to rule 23, TS(i) is simply the total profit (the area between the supply and demand curves to the left of the intersection of supply and demand) of all the N(i) DA games in environment E(i) (recall an environment is simply a complete specification of all game parameters, nbuyers, nsellers, ntokens, etc. as detailed in rule 17).

Note carefully that TS(i) is distinct from TP(i), the actual total token trading profits earned in E(i). Theoretical surplus TS(i) will be different from actual trading profits TP(i) if trade in the DAT is less than 100% efficient.

Also, in rule 23 you say "...c(i)...will be set to satisfy..."; which seems to mean that A(i) is pre-determined, and then, given TS(i), we have c(i) = A(i)/TS(i). Then you say "...c(i) will thus determined a priori." What is pre-determined c(i) or A(i)?

In a sense, both c(i) and A(i) are pre-determined.

A(i) is determined by the organizers before the start of the tournament, reflecting the share of the $10,000 prize money that we want paid out in the N(i) games played in environment E(i).

c(i) is determined by the formula c(i)=A(i)/TS(i) as described in Rule 23. TS(i) is "pre-determined" in the sense that its values are implied by parameter settings for environment E(i) set by the organizers, and the outcome of the N(i) draws of the random token generator.

However one might argue that TS(i) is not pre-determined since draws of the random token generator are stochastic (indeed, they are IID draws from a multivariate distribution specified in Rule 25). That's why we can't guarantee paying out exactly $10,000 with probability 1: the conversion ratio is a random variable. However by invoking the Law of Large Numbers, we can choose N(i) large enough that the variance in c(i) is arbitrarily close to zero, i.e. c(i) converges to the constant c(i)=A(i)/EP(i) with probability 1. EP(i) is the expected surplus in a single game played in environment E(i), a function of the parameters set by the organizers for E(i) such as the parameters of the random token generator (RAN1,...,RAN4) defined in Rule 25. EP(i) will also equal total expected trading profits in a single game if trade is 100% efficient.

[A sketch of the above result goes as follows. Suppose we vary the dollar amount paid out in environment E(i) as a function of the total games played N(i). Thus, let A(i,N(i)) be some rule which specifies the total dollar payment to the N(i) games played in environment E(i). Assume that as N(i) tends to infinity, A(i,N(i))/N(i) converges to a constant A(i) in the interval [0,10000].

Since the realized surplus in the N(i) games are IID draws, the Law of Large numbers guarantees that TS(i)/N(i) converges to EP(i) with probability 1. It the follows immediately that

$$c(i)=A(i,N(i))/TS(i) = [A(i,N(i))/N(i)]/[TS(i)/N(i)]$$

converges to the constant A(i)/EP(i) with probability 1.]

The bottom line is that by choosing N(i) large enough, we can guarantee that the conversion ratios are constants whose values are implied by the organizers' choice of the parameters for E(i).

Total dollar payments in the DAT will therefore differ from $10,000 only to the extent that 1) the realized conversion ratio c(i) (computed by Monte Carlo simulation over N(i) draws of the random token generator) deviates from its "theoretical expected value", A(i)/EP(i), and 2) actual trade in DA games is less than 100% efficient. The latter effect causes actual dollar payments to be lower than the intended amount of $10,000, whereas the former effect could cause payments to be higher or lower than intended. We expect that by choosing N(i) sufficiently large we will actually pay out within a few percent of that amount, however.

There seems to be a fair amount of interest at our University in the double auction game. I was one of the first to get interested, and the one that ftp'd all of the documentation and files so I am by default in charge of organizing a local tourney here. Having only had documentation on-line for a couple of days has generated interest from about a dozen people in three different departments. In light of this I have a couple of questions:

Glad to hear of the interest. If you've got a lot of people interested you might want to print up some manuals, or order some from Santa Fe. The ones from Santa Fe are a bit more readable than pure ascii—they were dressed up with a word processor.

As I will be running a local game, are they any pointers that you might be able to give me to make it run smoothly.

I've nothing very specific to say at present. You might want to construct a local token exchange, so people can connect via the network across the university. But that means buying the networking stuff ($200) from Santa Fe. If you're going to accept submissions of programs, you need to decide on your rules, parameter values, etc beforehand, and announce them. See our "rules" chapter. Unless you assign very similar token values to each buyer, and similarly for sellers, you won't be able to determine winners from single games. The variance is too large. So you'll need to run many games and average. I strongly suggest one player = one directory. Sub-directories of the monitor's directory are efficient. Use pipe-based communication if possible.

There seems to be some concern among the people here that the 100 person limit might restrict entries which have been developed over a long period of time and thus not submitted until near the end of the signup period. Do you think this is going to be a problem, and is there a way that we might guarantee that the two best entries from here be entered into the tourney, even though we may not know whose entries these are, or the manner in which the programs operate.

As you'll see from the rules, we'll have some reserved slots for this situation. We can reserve you one or two slots for unknown programs if we're pretty sure they'll actually materialize. Then further entries can go into the first-come/first-served category. We'll eliminate duplicates if your winners are already in the first/first bin.

As an interested party (and possible player) of the DAT, I was wondering if the folks at Santa Fe are planning on making any distinctions between types of players based on the level of pre-programmed strategy.

Some computer players may be very sophisticated and operate with a variety of programmer selected strategies. Other players may be intentionally designed with initially simple

strategies with the hope that learning algorithms will develop strategies on there own. The former may be likely to defeat the later soundly (maybe not?), but it seems to me the interest of the programmers who design the players are different, so at least some designation of the different approaches may be useful from an experimental point of view.

Maybe the following designations would be useful:

1) Fixed strategy player.

2) Adaptive player (varying parameters of a mostly fixed strategy).

3) Learning player (initialized with very little strategy, with learning algorithms attached).

Although the interactions of all three sorts of players in the same auction are interesting, tournaments consisting of players of each type may be exciting as well.

I would like to hear thoughts on these issues from other potential players and the auction organizers.

We recognize that there will be a wide variety of strategies playing in the DAT. It's difficult, however, to define what "adaptive" strategies are and how they differ from other types of "learning algorithms". We feel that random matchings are appropriate for the tournament to give all entrants an equal chance. However in our investigations after the tournament we will be able to conduct precisely the kinds of experiments you suggest.

In advance of the tournament we can also have "trial heats" for certain classes of games and player types. We will be announcing such "trial heats" shortly. We invite you (or anyone else on dat-list) to suggest interesting scenarios for the trial heats, at pre-announced times on the SFTE.

Query for the DAT organizers:

Do you consider it kosher to submit players which attempt to cooperate with other players? I can see a range of such strategies:

1	*one which tries to avoid confrontation*	*[obviously OK]*
2	*one which tries to communicate with other players through the size of bid jumps, without prearrangement*	*[probably OK]*
3	*one which communicates using prearranged signals communicated in the low-order bits of actual bids*	*[maybe OK; what's the ruling?]*

There are some interesting strategies centered around communication models. This is reminiscent of bridge bidding. It would be helpful to know which strategies you might encourage and which you might exclude.

According to our rules, all such strategies are OK. For example, we will allow strategies which attempt to "collude" via "handshaking" through prearranged bid and/or offer signals at the beginning of a DA game. This implies that certain entrants may wish to "collude" in their design of trading algorithms in advance of the tournament. Our rules do not prohibit such collaboration subject to the requirement of at most one trading program per entrant or entrant group.

You should be aware, however, that in the tournament programs will be randomly selected to play in DA games so there is no guarantee that you will have anyone to collaborate with. Furthermore, in the tournament players will not be explicitly identified by player numbers, they will be set to 0. Thus, your programs will have to send special messages to find out if potential collaborators are present.

Enforcement of 1 entry rule gets murky when several entrants are participating in the design of trading programs: are two such entrants submitting 1 or 2 programs to the DAT? In practice

we can't prohibit collusion either in the DAT itself or in the design phase in advance of the tournament. The one entry rule as we see it means that each entrant can submit at most one entry to the tournament. If the entrant worked on several trading programs, they will have to choose only one for purposes of collecting cash payments in the tournament.

The one/entry per person rule was designed to keep one person from hogging the $10,000 prize money by flooding the tournament with essentially duplicate trading programs.

However we shall absolutely prohibit any attempts to violate privacy of information, or to by-pass the monitor. Any program which tries to query the system and read other player's private game or message files will be subject to immediate disqualification. If some groups of players wish to share their private information, perhaps through clever messages concealed in patterns or values of their bids an offers, they can do so. However they must always pass their information along public communication channels, i.e. via the monitor. Players CANNOT write private messages directly to each each other. This would be illegal collusion akin to stealing other players' private information. Be assured that we will deal with these types of illegal acts just as harshly as the SEC dealt with Ivan Boesky.

Perhaps you'd like to know about a small loophole in the rules for the DAT. It allows a group of partners, if they pool their efforts, to achieve mega-profits, by having one partner lose large amounts of DAT-money, and the other win equal amounts. It takes advantage of the fact that you can't ask players to write you a check if they make negative profits.

This isn't too much of a worry; such a scheme is unlikely to actually take place, and is pretty easily preventable. But you should realize that such maga-loss behavior is not just aberrant, but may be an effort to garner more profits for a consortium.

Anyhow, the details of the scheme are as follows:

A and B are partners. In game G they discover that A is a buyer and B is a seller. They have agreed to bid as follows in such a situation: The seller offers his tokens at min-price, and the buyer bids at max-price. The seller accepts, making a profit of (max-price - cost(i)) for each token i. The buyer's profit on each is a large negative amount, (redemption(i) - max-price).

If this happened often enough that one of the partners achieved negative total dollar winnings, he has succeeded in transferring money to the people with whom he made the "sacrifice trades". One way to improve the ratio of games in which this can occur would be to have a team of partners, half of whom are designated losers, and the other half of whom are designated winners. If the losers see winners on the other side, they sacrifice.

OK, OK, maybe I'm a little too into conspiracies...but you may want to monitor for large losses, and note if they benefit some set of players....

Although our rules don't prohibit collusion, we don't allow strategies that make deliberate losses, see rule 26 of chapter 7. Thus, we have forseen and outlawed the type of "loophole" you describe.

I have three questions about the rules. They concern the tournament itself, not double auction mechanics.

1. I am unclear about the extent of participation by the organizers. For example, will entries submitted by organizers also be rewarded from the $10k pool, or are those entries exempt? If all 30 such slots are used, then an entry would be rewarded 10000(n/(n+30)), 1<=n<=70, assuming all programs earn equal profit. Thus, the prize money to all the non-organizers would be between $323 and $7000 total.*

The three organizers (Miller,Palmer,Rust) have declared themselves ineligible to participate in the tournament. The 30 reserved slots will be allocated to individuals who the organizers believe to be strongly committed to producing a "serious" and/or "interesting" trader program. Thus,

assuming the 100 entry limit is binding and an equal distribution of payouts, the 30 reserved slots would account for $3,000 of the $10,000 prize money and the other 70 would be paid approximately $7,000. Currently only 5 or 6 reservations have been made out of the total of 30 potential slots. If we do not reserve all 30 slots, the remaining slots will be added to the 70 "first come, first served" slots as described in rules 1 and 2.

2. *In the same vein, will organizers be examining the code of the outside entries prior to the playing of the official games? If organizers are vying for the $10k, then the ability to inspect the code of the other entrants ahead of time would seem to be a distinct advantage to the organizers.*

This issue is moot since the organizers won't be participating in the tournament. We guarantee the confidentiality of source code submitted before the March 1, 1990 deadline. Entries will be examined as they are received to verify that the source code conforms to the rules. We will compile the player programs to see if they run the SFI Sun-4. Programs that violate the rules, or which fail to compile, or which compile but have obvious bugs will be returned to entrants for 1 opportunity of revision. Entrants will then have until March 1, 1990 to resubmit a revised strategy that fixes the problems we discovered in the initial compilation. Thus, programs received exactly on March 1, 1990 will have essentially no opportunity to be revised. It is therefore a good idea to submit your code somewhat in advance of the deadline to ensure sufficient time to make final modifications in case some problems arise.

Our guarantee of secrecy also applies to any programs submitted to be regular local players on the Santa Fe Token Exchange in advance of the tournament. We will compile the submitted code, but will guarantee its confidentiality until the tournament is run in March 1990.

3. *Last, is it possible to secure a slot prior to submitting the code? Assuming I decide to continue with this, I would like to be able to work on my code until close to the deadline, then send it in. Otherwise, the longer I work on my code, the more risk there is of missing the boat, which would be frustrating to say the least. Would it be possible to allow paying the $10.00 early to secure a spot, and then allow forwarding the code at a later date?*

The only way to guarantee being chosen as one of the 70 "first-come, first-served" entries is to submit a program, entry form and $10.00 fee before 70 other people have signed up. According to rule 1, once you submit an entry (which includes the source code) you will be allowed at most 1 opportunity to revise it, and those revisions can only be minor revisions that are necessary to allow the program to run on the SFI Sun-4, or possibly to correct minor violations of the rules. Thus, you won't be allowed to just submit an entry form and the $10.00 fee and continue to develop your code. This obviously creates a trade-off: submitting early guarantees a slot but reduces your development time, while submitting towards the March 1 deadline gives you more development time, but increases the risk of missing the boat. While we don't like to impose this risk on potential entrants, none of the alternatives seem satisfactory. The basic problem is to limit the number of programs so that the winnings per entrant is reasonably high. We want to attract the best possible entries and we feel the combination of reserved slots and "first-come, first-served" slots is the best compromise to meet this objective. You can reduce your risk somewhat by keeping close tabs on the "entries" file in the /pub/dat directory (available via anonymous ftp or via phone, fax, or U.S. mail. We will also update the entries file to keep a current list of the total number of reserved slots that have been assigned so far.