

WOODSTOCK  
A Wireless Stock Trading Game for  
Windows CE™

Luis J. Vega  
Department of Computer Science  
Brown University

Submitted in partial fulfillment of the requirements for the Degree of Master of Science  
in the Department of Computer Science at Brown University

---

Signature (Prof. Maurice Herlihy, Project Advisor)

---

Date



# Acknowledgements

I would like to thank my project advisor, professor Maurice Herlihy, for his help in all areas of this project. I would also like to thank Mike Coglianesi and Benety Goh for their help with wireless Aleph; and Ravi Jeyaratnam for lending me the machines for the user study.

Also, thanks to the following people for being part of the user study: Daniel Acevedo, Mike Coglianesi, Benety Goh, Ravi Jeyaratnam, Luis Ortiz, Changhee Pyo, Charles Thompson, and Shaoqing Shi.



# Table of Contents

1. Introduction .....	7
2. User-level Functionality.....	8
3. Client and Server Functionality.....	10
3.1 The Woodstock Server .....	10
3.2 The Woodstock Client.....	11
4. Client/Server Communications.....	13
4.1 Wireless Connection to the LAN.....	13
4.2 Communication Managers.....	13
4.3 Booting the Client.....	14
4.3.1 Finding The Server.....	14
4.3.2 Initial Client Boot .....	14
4.3.3 Subsequent Client Boots.....	14
4.3.4 Startup Protocol.....	15
4.4 Buying and Selling.....	15
4.4.1 Buy/Sell Protocol.....	16
4.5 On-line vs. Off-line .....	17
4.6 Multicasting Stock Prices .....	17
4.7 Multicasting the End of Game.....	18
5. Performance.....	19
6. Issues, Solutions, and Work-arounds .....	20
6.1 Wireless Connection to the LAN.....	20
6.2 Changing IP .....	20
6.3 Suspending the Client .....	22
6.4 Vulnerability.....	23
6.5 Web Site Access.....	23
6.6 Client/Server Communications.....	23
6.7 UI Issues .....	24

7. User Study .....	26
7.1 Software Problems .....	26
7.2 Functionality .....	26
7.3 Performance .....	27
7.4 Connectivity .....	27
8. Conclusions .....	28
Appendix A: Woodstock Files .....	31
Appendix B: User Study Questionnaire .....	33
Appendix C: Woodstock Screen Shots .....	35
Appendix D: Woodstock Web Site .....	39

# 1. Introduction

Woodstock (Wireless Object Oriented Stock Trading On-line Commerce Kit) is a stock trading game. Players are given \$10,000 virtual dollars which they can use to buy stocks. The stock prices displayed are actual prices of the NYSE and NASDAQ markets (although they are only updated every 30 minutes). When the game is over, the user with the largest profit is the winner.

Woodstock was developed for the Windows CE operating system. It is a client/server system in which clients communicate with the server via a wireless network. The system is written using the Java programming language and it uses the wireless version of the Aleph toolkit<sup>1</sup>. Woodstock's source code is located in `/pro/aleph/sandbox/ljv/aleph1.1.0-ce/aleph/woodstock`.

This paper is written for computer scientists who want to learn about the inner workings of Woodstock and the issues encountered during its development. The author assumes that the reader is familiar with the Java programming language, the Aleph toolkit, and the Windows CE operating system. Section 2 of this paper describes Woodstock's user level functionality. Sections 3 and 4 focus on Woodstock's system level functionality and provide implementation information. Section 5 provides quantitative measures of the system's performance. Section 6 describes issues encountered during the creation of Woodstock. Section 7 presents the results of a user-study conducted to test the system and to determine its usability. Section 8 concludes.

---

<sup>1</sup> The Aleph toolkit is a collection of Java packages which supports the construction of distributed systems.

## 2. User-level Functionality

Players are presented with a list of 20 stocks to choose from and they can buy/sell shares at any time. The stock prices displayed are the actual prices of the NYSE and NASDAQ markets. These prices are updated every 30 minutes during stock trading hours (i.e., Monday through Friday 9:30a.m. - 4:00p.m.). When the game is over, the user with the largest profit is the winner.

Woodstock is a Client/Server system. Buy/Sell transactions are requested by the Woodstock Client (running on the CE device) and executed in the Woodstock Server. The client operates on two modes: On-line (connected to the Server) and Off-line (disconnected from the Server). Woodstock displays the current mode in the bottom right corner of its main window<sup>2</sup>. The main window consists of 6 tabbed panels. The "Stock Prices" panel contains stock price information for all stocks that the user can buy or sell. To buy or sell a stock the user clicks the "buy" or "sell" button for that stock. The "My Portfolio" panel contains information about the user's account such as the stocks that he owns and the profit/loss made on these stocks. Profitable stocks are shown in green. Losing stocks are shown in red. The "Trade History" panel contains information about all successfully completed buy/sell transactions. The "Failed Transactions" panel contains information about buy/sell transactions that have failed. The "Pending Transactions" panel contains information about pending transactions. The "About" tab contains general information about Woodstock.

To buy a stock the user enters the number of shares he wants to buy and a price below which he is willing to buy these shares. Conversely, to sell a stock, the user enters the number of shares he wants to sell and a price above which he is willing to sell these shares. A transaction is considered pending if (1) the user requests a buy/sell transaction but it has not yet been sent to the Woodstock Server (most likely because the Woodstock client is Off-line); or (2) the user requested a buy/sell transaction and it was sent to the

---

<sup>2</sup> The reader is referred to Appendix D which contains screen shots of Woodstock's user interface.



Woodstock Server but the Woodstock client has not yet received a confirmation about this transaction from the Server. Pending transactions that were not sent to the Woodstock server because the user was working Off-line are sent to the server the next time a connection to the server is established. The only way a connection to the server can be reestablished is by exiting the Woodstock client application and launching it again. This is a limitation of the system that is explained in section 6.2.

## 3. Client and Server Functionality

### 3.1 The Woodstock Server

The Woodstock Server provides information for, and handles trading transactions from, the Woodstock clients. A server thread gathers stock prices every 30 minutes (it only gathers the stock prices when the stock market is open since it is unnecessary to gather prices when the market is closed) and multicasts these prices to all clients. Stock prices are gathered by html-scraping [www.pcquote.com](http://www.pcquote.com). This web site contains free stock price information which is delayed by 20 minutes. This site was chosen over other possibilities because (i) the price of up to 20 stocks can be obtained with a single query and (ii) this query can be easily formulated programmatically and included as part of the URL. For example, in order to retrieve the stock prices for America On-Line (AOL) and Lucent Technologies (LU), the Woodstock Server connects to [www.pcquote.com/stocks/quotedirect.php?ticker=AOL+LU](http://www.pcquote.com/stocks/quotedirect.php?ticker=AOL+LU) and html-scrapes the resulting web page.

The server is designed to run indefinitely. In this way, users can check their portfolio and trade stocks at any time. After the game is over, the Woodstock server remains running in order to allow users to check the final balance of their portfolio, but it doesn't allow users to perform any additional trading transactions.

The end of the game is determined as follows. At startup, the server is given the date and time in which the game should be ended via a command line argument. A server thread runs periodically, checking the current date/time against the given end of game date/time. When the current time is equal to, or later than, the given time, the Server notifies all (connected) clients about the end of the game.

The server stores account information for each of the game players. This information includes the account number, player's name and e-mail address, amount of "cash" available, and number and type of shares currently held in this account (e.g., 50 shares of

Lucent, 10 shares of AOL, etc.). The account information is stored in memory. It was also decided to store this information on disk. The disk copy is used in case the Server unexpectedly terminates or needs to be restarted manually. If this occurs, upon restart the server will read the account information from disk. If this information were not saved on disk, all account information would be lost due to a server crash or a manual restart.

The Woodstock Server was written in Java using JDK 1.1.x. and is approximately 3500 lines of code. It was designed to run on any workstation connected to the Brown CS Department's LAN.

### **3.2 The Woodstock Client**

End-users interact directly with the Woodstock client in order to buy/sell stocks. It maintains the following information in memory and on disk: (i) account information (e.g., account number, which stocks are owned by this account); (ii) stock prices; (iii) failed transactions; (iv) successful transactions; and (v) pending transactions. Items (iii) through (v) need to be stored on disk so that the information is not lost when the client is shut down. The stock prices (item (ii)) are read from disk only when the Client is working off-line since this information is retrieved from the Server when the Client is working on-line. The account information (item (i)) is always read from disk when the Client is booted (except for the first time the Client is started). If the client is on-line, the account number (stored as part of the account information) is used to obtain the account information from the server. If the client is off-line, the account information read from disk is displayed to the user. In this way, even though the client maintains a copy of the account information, the "master" copy is maintained by the server. A malicious user could modify the client copy (and give himself large amounts of money or large number of shares) but the master copy is the one used to perform all buy/sell transactions, defeating the malicious user's attempt. Furthermore, the client copy is replaced with the server's copy every time the client connects to the server or performs a buy/sell transaction.

The Woodstock client was written in Java using JDK 1.1.x and is approximately 5400 lines of code (not including the public domain UI libraries it uses). It was written to run on Sun's PersonalJava Runtime Environment 1.0 (which is a JVM for Windows CE). Since the target platform for the Woodstock Client is Windows CE, the UI code was written to look and behave as expected when run on this platform at the expense of unexpected look and behavior on other operating systems.

## 4. Client/Server Communications

All communications between the Client and the Server were written using the facilities provided by the Aleph toolkit. Specifically, Woodstock uses Aleph's direct message passing model and Aleph's event model.

### 4.1 Wireless Connection to the LAN

Before starting the Woodstock Client, users must decide whether to work on-line or off-line. If they want to work on-line, their CE device needs to be connected to the CS department's LAN via a wireless connection. In order to connect to the LAN, users must insert their LAN card. Upon inserting the LAN card, the CE device attempts to connect to the LAN and to obtain an IP address (using DHCP). Only after an IP address is obtained is the Woodstock Client able to communicate with the Woodstock Server.

### 4.2 Communication Managers

The Woodstock Client uses Aleph's UDP communication manager. The Woodstock server uses a custom communication manager, `WoodstockCommManager`, which is based on Aleph's UDP communication manager. `WoodstockCommManager` differs from Aleph's UDP communication manager in the following two ways. First, `WoodstockCommManager` was written to always use port 3030<sup>3</sup>, unlike Aleph's UDP communication manager which uses whatever port is chosen by the creation of a new `Java DatagramSocket` object. Second, `WoodstockCommManager` was designed to periodically clean up its hash table of connection objects, whereas Aleph's UDP communication manager doesn't perform this clean up. Given that the server was designed to run for an arbitrary period of time, it was decided that the hash table of

---

<sup>3</sup> this value is currently hardcoded but it could be easily changed to be read from a file or an argument to the Woodstock server.

connections should be cleaned up periodically (by default every 40 minutes) so that it wouldn't grow without bound. A server thread performs this clean up by periodically pinging all clients and removing the hash table entry for those clients who can't be pinged.

## **4.3 Booting the Client**

### **4.3.1 Finding The Server**

The Server can be run on any workstation. As mentioned above, it will always use port 3030 to communicate with the clients. The clients are hardcoded to look for the server in port 3030. The IP of the machine running the Server is not hardcoded and it is passed to the Woodstock client as an argument.

### **4.3.2 Initial Client Boot**

The first time a Woodstock Client is started, it prompts the user for his name and e-mail address. It then pings the server (using the UDP communication manager's ping method). An unsuccessful ping takes place if the client machine is not connected to the wireless LAN (i.e., the user didn't insert the LAN card or, if the user did insert the LAN card, an IP address was not successfully obtained). If the ping is not successful the user is asked to connect to the LAN and the application is exited. If the ping is successful, the start-up protocol (described in section 4.3.4) is initiated.

### **4.3.3 Subsequent Client Boots**

When the Client is started other than the first time, it pings the Server to determine if the server is reachable. If the ping is not successful it prompts the user whether to work off-line or to exit the application. If the ping is successful, the start-up protocol (described below) is initiated.

If the user chooses to work offline, the Woodstock client retrieves the account and stock information from its local files in order to display this information as it existed the last time the user used the system. Whether working on-line or off-line, the client reads from disk any pending transactions, successful transactions, and failed transactions in order to populate their corresponding displays.

#### **4.3.4 Startup Protocol**

This protocol was designed so that it could be used for both the initial Client boot and subsequent Client boots. The client starts the protocol by sending message `GetAccountAndStocksInfo_request` to the Server. If the client is being booted for the first time this message contains a flag asking the server to create a new account. Otherwise, this message contains the account number of the client (which is stored on the client's disk). The Server replies with message `GetAccountAndStocksInfo_response` which contains two objects: one containing the client's account information and the other containing the most recent stock prices stored in the Server. After the client gets this response (it times out and informs the user if it doesn't get this response) it sends message `RegisterEventListeners_request` to the Server. The server replies to this request with message `RegisterEventListeners_response`. This message contains two aleph event objects: the `UpdateStocksInfoEvent`, which is used by the server to multicast stock prices, and the `EndOfGame` event, which is used by the server to multicast that the end of the game has been reached. Upon receiving the `RegisterEventListeners_response` message, the Client registers listeners for these events. Lastly, if there are any pending transactions that need to be sent to the server, the client sends them using the Buy/Sell protocol described in section 4.4.1.

### **4.4 Buying and Selling**

When the user requests to buy or sell shares of a particular stock a pending transaction is created, stored in memory, and saved to disk.

If the client is working on-line, it pings the server. If the ping is successful, a status of "sent" is given to this transaction and it is sent to the server using the Buy/Sell protocol described below. If the ping is unsuccessful, the client changes from working on-line to working off-line.

If the client is off-line, a status of "unsent" is given to the buy/sell transaction and a message box informs the user that the transaction will be queued until a connection to the server is established. Unsent pending transactions are sent to the server the next time the client connects to the Server. Sending pending transactions upon reconnection<sup>4</sup> is simple: the file containing the pending transactions is read from disk and transaction requests that don't have a "sent" status are sent to the server using the Buy/Sell protocol.

#### **4.4.1 Buy/Sell Protocol**

This protocol was designed so that it could be used for both buy and sell transactions. The client starts by sending the message `BuySellTransaction_request` to the Server. This message contains the type of transaction requested (buy or sell) and the transaction parameters (e.g., which stock to buy/sell, how many shares to buy/sell, price below which to buy/price above which to sell). Upon receiving this message, the server determines whether the transaction can be performed by checking conditions such as: does the user have enough cash to buy the requested number of stocks? is the current stock price below (or above) the price requested by the user? does the user own at least as many shares of the stock that he requested to sell? If the transaction can be performed, the Server updates the user's account information to reflect the buying/selling of the requested shares. If the transaction can't be performed, it is considered to have failed. Whether the transaction fails or succeeds, the Server sends a `BuySellTransaction_response` message to the client informing it of the completion status of the requested trading transaction. Upon receiving this message, the client removes the transaction from the in-memory object (and its associated disk file) that stores the list of pending transactions. If the transaction was successful, the client adds a corresponding

---

<sup>4</sup> recall that a reconnection can only occur when the Client is shutdown and then started.



entry to the in-memory object (and its associated disk file) that stores the trade history and updates the relevant displays (e.g., the My Portfolio and Trade History panel). If the transaction was not successful, the client adds a corresponding entry to the in-memory object (and its associated disk file) that stores failed transactions, informs the user about the failed transaction, and updates the relevant displays (i.e., the Failed Transactions panel).

## **4.5 On-line vs. Off-line**

During the startup protocol, the client pings the server. If the server can't be pinged, the user is given the option to work off-line or to exit the application. If the user chooses to work off-line, the client remains off-line and does not attempt to reconnect to the server (this is discussed further in section 6.2). If the server can be pinged at startup, the client starts in on-line mode. Whenever the user requests a buy or sell transaction, the client pings the server before attempting to send it a message requesting the transaction. If the ping is successful, the client remains on-line. If the ping is unsuccessful, the client changes to an off-line status and it can never change back to an on-line status. In short, while the client is running, (1) it can change from on-line mode to off-line mode if the server can't be pinged when a user requests a buy/sell transaction; and (2) it can never change from off-line mode to on-line mode.

## **4.6 Multicasting Stock Prices**

A server thread retrieves stock prices from a web site every 30 minutes and multicasts these prices to the clients. The multicast is done by signaling the `UpdateStocksInfoEvent` (this is an Aleph event) and including as part of the signal the Java object containing the new stocks information. Upon receiving the signal, the client sets its memory and disk copies of the stock information to be this new object and updates its displays. Note that all clients register a listener for the `UpdateStocksInfoEvent` during the startup protocol.

## **4.7 Multicasting the End of the Game**

The server multicasts the end of the game by signaling the EndOfGameEvent (this is an Aleph event). Note that all clients register an EndOfGameEvent listener during the startup protocol.

## 5. Performance

This section provides general information on Woodstock's performance.

The Woodstock Client's startup time (from the moment the user double clicks the Woodstock icon to the time the main screen is displayed) is approximately 38 seconds. Most users describe this startup as "acceptably slow" (see section 7: user study). The time to buy/sell a stock (from the moment the user hits the "enter" button up to the moment he is given visual feedback on the transaction) is approximately 12 seconds, 3 of which are due to the communications overhead between the client and the server. Most users found this 12 second delay time to be "just fine" (see user study).

The Woodstock Server takes approximately 1.25 seconds to connect to and retrieve the html from pcquote.com (this measure doesn't include the time to html-scrape the web page). Parsing the html to retrieve the stock prices takes approximately 18 milliseconds. Multicasting the stock prices takes approximately 1.3 seconds (from the time the Server signals the event to the time the client receives the signal). A ping from the client to the server takes approximately 61 milliseconds (from the time the client pings the server until the client hears back from the server)

The aforementioned performance levels can't be significantly improved by algorithmic enhancements to Woodstock due to the fact that Woodstock's performance is governed by the speed of the processor being used, the wireless network, and the JVM on which it runs.

## **6. Issues, Solutions, and Work-arounds**

### **6.1 Wireless Connection to the LAN**

While developing and testing Woodstock, the author experienced a reoccurring problem with the wireless connection to the LAN. Often times, upon inserting the LAN card, the connection icon (in the task bar) would indicate that the CE device was successfully connected when it was not (e.g., the client was unable to ping a workstation known to be connected to the LAN). When this occurred, removing and re-inserting the LAN card did not solve the problem. The author found that removing the card, soft-resetting the machine, and then inserting the card solved the problem. The user study revealed that this was indeed a common problem.

### **6.2 Changing IP**

While the Woodstock client is running it is possible for the IP address of the machine in which the client is running to change. For example, if the user removes the LAN card, the IP address is set to 127.0.0.1. Currently, there is no way for Woodstock to detect that the IP address changed. The problem arises because Java caches the IP address of the machine in which it is running. If the address changes, the cache is not updated and calls to any Java methods that retrieve the IP address (e.g., `InetAddress.getLocalHost()`) retrieve the cached (outdated) address.

Consider the scenario in which Woodstock is running using IP address A. Then the user removes the card. The IP address is now 127.0.0.1. The user then re-inserts the card. The IP address is now B. The user buys/sells a stock. Since the Woodstock client can ping the Server, it sends it a message requesting the buy/sell transaction. Embedded in this message is IP address A. The server processes the request no different than it would process any other request and sends a message to the client to inform the completion status of the request. The problem is that the server sends this message to address A (the

old address) instead of address B (the new and correct address). We refer to this problem as the *changing IP problem*. The scenario where the user removes the LAN card is not the only one in which the changing IP problem can occur. It can also take place when the machine goes out of the range of the wireless LAN and then gets back into the range.

If the Woodstock client is working on-line, there is no way for the client to determine that the problem scenarios have occurred, and, therefore, there is no way to prevent the problem from happening. However, the scenarios described are unlikely due to the fact that the protocol for connecting to the LAN (DHCP) tries to use the same IP address that it previously used. The user study provided evidence that this problem is unlikely since the problem was never encountered during the study.

If the client is working off-line, there is a way to prevent the changing IP problem from happening, although at the expense of some end-user functionality. To prevent the problem, once the client is working off-line, it doesn't attempt to automatically get back on-line. Hence, the only way that a user can connect to the server once the client is off-line is to close the Woodstock client and launch it again. Ideally, instead, the client would just reconnect automatically without user intervention. However, an automatic reconnect could cause the changing IP problem. To see how the problem can occur, consider the following scenario. The client is working off-line. This means that in a previous attempt to ping the server, the ping was not successful. This could be because the user started Woodstock without using the LAN card or because the user is out of range of the wireless network. Then, the user inserts the LAN card or enters the range of the wireless network. The CE device gets an IP address, B, and this address is different from IP address A, which is the one it had before (if the LAN card was not inserted A = 127.0.0.1). At this point the Server is reachable, so the next time a buy/sell transaction takes place the client pings the server and since the ping is successful, the client decides that it has reconnected to the server and starts working in on-line mode. At this point, the client machine's real IP address is B but Woodstock "thinks" (as explained in the previous paragraph) that its address is A. Therefore, in order to prevent the problem, even if the server can be pinged, Woodstock remains working off-line.

An ideal solution to the changing IP problem would be to detect when the IP changes. This could be done in various ways. Some ideas are: to override the Java implementation of methods such as *GetLocalHost()* to not cache the IP address; to create a custom native method that obtains the IP address via system calls; and to have a native demon process (e.g., written in C++) that periodically gets the IP address and saves it to a file which is periodically read by Woodstock. The evaluation of these, and other ideas, is an area of interesting future work.

### **6.3 Suspending the Client**

Another issue encountered was in regards to the "suspend" functionality of Windows CE. Consider the scenario where the Woodstock client is on-line and the user suspends the CE device without closing the Woodstock application or removing the LAN card. When the user "unsuspends" the machine (turns it on) and it is still within the range of the wireless LAN, the actual IP address of the machine usually remains unchanged. Given that the IP address didn't change it should be possible for the user to keep working on-line and therefore Woodstock's functionality should be no different than what it was before the user suspended the machine. The problem is that Server might have dropped the connection due to connections clean-up (see section 4.2 for a discussion on this clean-up). If the server drops the connection, the next time the user requests a buy/sell transaction, the client pings the server and the server's communication manager, as expected, creates a new connection for this client. However, subsequent messages from the client never get to the server. Further analysis is needed to determine the exact cause of the problem. It is interesting to note that this problem never occurred during the user study. The study revealed that users always closed the Woodstock client before suspending their machines.

## **6.4 Vulnerability**

Woodstock is vulnerable to client and server crashes as well as network partitioning. It could be made more robust by implementing atomicity of its buy/sell transactions. This would be an interesting area of future research. Aleph's current transaction model could be extended to support atomic distributed transactions (among different Processing Elements) and Woodstock could be modified to use these Aleph transactions to implement its buy/sell transactions.

## **6.5 Web Site Access**

The Woodstock Server accessed a web site every 30 minutes to retrieve stock information. On average, it took approximately 1.25 seconds to connect to and retrieve information from the web site (not counting html-scraping time). However, the user study revealed that sometimes the Server would wait a seemingly infinite amount of time when attempting to connect and retrieve data from the web site. Further analysis needs to be performed to determine the exact cause of this problem. A simple solution would be to add a time out period after which the data retrieval is cancelled and then restarted. We note that this problem can be recreated, but there is no certainty as to when it will occur. That is, the Server needs to be running for a few days before the problem occurs. During the user study the problem was "fixed" by manually stopping and restarting the server.

## **6.6 Client/Server Communications**

The user study revealed that sometimes during the startup protocol, the client would ping the server and the server's communication manager would create a connection object for the client (as expected), but the ping would be unsuccessful (i.e., it timed out). The author found that this problem is not unique to the wireless environment as it was initially thought (i.e., running a Woodstock client on a workstation yielded the same problem).

Further analysis is needed to determine the cause of this problem. We note that, similar to the problem discussed above, this problem can be recreated but there is no certainty as to when it will occur. That is, the Server needs to be running for a few days and the client needs to connect and disconnect a few times before the problem occurs. This problem caused users to not be able to connect to the Woodstock Server, but after trying to connect again for a few times (by restarting the client application), users were able to connect.

## 6.7 UI Issues

The UI was developed using two libraries obtained from the World Wide Web<sup>5</sup>. These libraries were initially tested to make sure they looked and behaved as expected when run on Windows CE. The UI was coded on Windows 95 as the look and feel of a Java application run on Windows CE is very similar to the look and feel when run under Windows 95. One issue encountered was that the mouse-clicked AWT event was not triggered most of time. Due to this problem, a user had to tap on a UI button multiple times before the expected behavior was achieved. The author noted that a mouse-pressed event was always triggered when the user tapped the screen. Hence, the solution was to always use the mouse-pressed event instead of the mouse-clicked event. A second issue was that a key-up event was being triggered when it shouldn't have been (e.g. when the user never pressed a key). The work-around was to remove the logic for handling this event from the code (this logic was not being used by Woodstock but it existed as part of one of the libraries used). A third issue was that mouse-pressed events were being triggered on the incorrect AWT components. Specifically, when tapping on a table's header component<sup>6</sup>, a mouse-pressed event was triggered in the table's data component. Conversely, when tapping on the table's data component, a mouse-pressed event was triggered in the table's header component. The problem was resolved by looking at the x

---

<sup>5</sup> these libraries were (1) Table Bean (from <http://www.datacomm.ch/tstuder/resources/index.html>) and (2) l2fprod (from <http://members.nbc.com/l2fprod/html/packages.html> )

<sup>6</sup> The display tables that are extensively used in Woodstock are implemented using one AWT component for the table header and a different AWT component for the table data.



and y coordinates of the mouse-pressed event and using them to infer the actual component on which the tap occurred. Note that these issues were all specific to Windows CE (and Personal Java) as the same sequence of events that caused the incorrect behaviors in Windows CE worked as expected in Windows 95.

## **7. User study**

Eight users played the Woodstock game for a period of two weeks. The purpose of the user study was twofold: to test the system and to determine its usability. The author periodically monitored the Server's status during the test to gather information about its behavior. Each user was given a questionnaire to fill out during the game and/or after the completion of the game, but only 7 users completed it. Appendix C contains this questionnaire. The author maintained verbal communication with the users during the testing period in order to gather feedback in addition to that in the questionnaire. Despite experiencing some problems (described below), all users were able to successfully use the application over the two week period.

### **7.1 Software Problems**

On the client side, one user reported that the splash screen sometimes remained displayed for a "long time" after exiting the application when a connection to the server couldn't be established at startup. One user also reported that, on one occasion, the stock prices stored in the client's disk (which are displayed when the client is working off-line) were not as current as they should have been (they should be as current as the most recent stock prices obtained from the Woodstock Server). Further analysis needs to be done in order to find the exact cause of these problems. On the server side, two issues were found which were described in sections 6.5 and 6.6.

### **7.2 Functionality**

All users found Woodstock to be easy to learn and found the buying/selling process intuitive, although two users initially thought that when buying/selling a stock Woodstock would wait until the stock had reached the requested price before performing the transaction. All users commented positively on the "look" of the UI and its ease of use. Most users found this "look-and-feel" to be what they liked most about Woodstock.

As for what users disliked the most, a majority of users mentioned the initial triple screen drawing of the tab displays<sup>7</sup>.

All users agreed that the information displayed was useful (i.e., none of the information was unnecessary). When asked for additional information that they would have liked to see, most users wanted more stock data (e.g., stock prices at closing time from the previous day, a plot of stock prices). As far as additional functionality, the two most requested items were: (1) Woodstock should go from off-line to on-line automatically (without having to restart the application) and (2) when buying/selling a stock, the user should be able to ask Woodstock to wait until the stock reaches the requested price before performing the transaction.

### **7.3 Performance**

Six users found the performance when buying/selling stocks to be "just fine" and one found it to be "acceptably slow". Five users found the performance when starting up Woodstock to be "acceptably slow" and two found it to be "unacceptably slow". As far as navigating through the tabbed panels, five users found it acceptably slow and two users found it to be just fine. Ideally, the system's performance would be perceived as "just fine" by all users. These results indicate that, even though the performance is not ideal, most users find it to be acceptable which suggests that the system is usable from a performance standpoint.

### **7.4 Connectivity**

As mentioned in section 6.1, the study revealed that the problem of not connecting to the LAN even after inserting the LAN card was common (all users experienced it). The author found this out via verbal communication with the users. The user test also revealed

---

<sup>7</sup> the author was not able to fix this problem despite investing significant time and effort. The problem seems to be related to the way in which AWT Paint messages are generated and processed in one of the public domain UI libraries used in Woodstock.

that even if the client machine was connected to the LAN, sometimes a connection to the server couldn't be establish (this is a problem with Woodstock that is explained in section 6.6). The author found this problem by monitoring Woodstock's behavior during the user study. The questionnaire failed to make the distinction between not being able to connect to LAN vs. not being able to connect to the Woodstock Server and therefore it wasn't very useful in obtaining information about the ability to connect to the server.

As far as maintaining the connection to the server (i.e., being able to ping the server before performing a buy/sell transaction), most users never lost the connection, one user lost the connection most of the time and one user lost it some of the time. These results are in accordance with the fact that radio frequency based wireless networks are known to be subject to interference.

## 8. Conclusions

In discussing the conclusions, it is helpful to decompose Woodstock into two different (yet related) levels: the end-user stock trading game and the technical machinery behind this end-user functionality.

From the standpoint of the stock trading game, the user study suggests that Woodstock's functionality and user interface are very effective in allowing users to play the game. The study also suggests that, even though the performance is not perceived as ideal, the current performance levels are not deterrents to the use of the application.

From the technical standpoint, the design, development, and testing of Woodstock was found to be a valuable experience in the creation of client/server systems in general, and in the creation of systems written in Java using wireless Windows-CE based clients in particular.

The issues found involving atomicity of transactions, handling Windows CE's suspend mode, and handling an IP address that can change while the client is running, are common to any distributed wireless system using Windows CE based clients. The study of these issues is applicable to a wide range of projects and are considered by the author to be important and interesting areas of future work.



# Appendix A: Woodstock Files

The following is a list of the files used by Woodstock.

## Woodstock Server

stocksinfo file - contains price information for each of the stocks available for the user to transact: the current stock price, the price at which the stock started when the market was opened, the lowest price for the day, and the highest price for the day.

accountsinfo file - contains the account information for all client accounts. Each account contains an account number, user name, e-mail address, amount of "cash" available, total amount of cash invested, and the number and type of shares currently held in this account.

accountsbalance file - this is a human readable file which contains information about the client accounts. The accounts are sorted descendingly by profit made. This file is created when the end of the game is reached and is used by the Woodstock administrator to see the profits made by each of the Woodstock players.

hraccountsinfo file - this is a human readable file which contains information about the client accounts. It is updated every time a client account is modified (for example, when a buy transaction takes place). This file was created so that the Woodstock administrator can monitor the status of the accounts at any point in time.

## **Woodstock Client**

stockinfo\_client file - Conceptually, this is the client's copy of the server's *stockinfo* file.

accountinfo\_client file - Conceptually, this is the client's copy of the part of the server's *accountinfo* file that contains information for this client's account.

tradehistory file - contains the information needed for the trade history display.

pendingtrans file - stores the client's pending transactions (which are displayed in the pending transactions tab).

failedtrans file - contains the information needed for the failed transactions display.



# Appendix B: User Questionnaire

## Functionality

1. Circle your answer. On a scale from 1 to 5 how easy was it to learn how to use Woodstock? (1 = easiest, 5 = hardest) 1 2 3 4 5
2. Circle your answer. On scale from 1 to 5, how intuitive was the process of buying a stock? (1 = very intuitive, 5 = very unintuitive) 1 2 3 4 5
3. Circle your answer. On scale from 1 to 5, how intuitive was the process of selling a stock? (1 = very intuitive, 5 = very unintuitive) 1 2 3 4 5
4. Did the software behave unexpectedly when buying a stock? If so, how?
5. Did the software behave unexpectedly when selling a stock? If so, how?

## Performance

In the following questions please circle 1, 2 or 3.

1 = unacceptably slow (it is so slow that it is unusable)

2 = acceptably slow (it is slow but I can live with it)

3 = just fine

6. How was the performance when buying a stock? 1 2 3
7. How was the performance when selling a stock? 1 2 3
8. How was the performance when starting up Woodstock? 1 2 3
9. How was the performance when navigating through the tabbed panels? 1 2 3

## Connectivity

10. Circle your answer. When starting Woodstock and expecting it to connect to the Woodstock Server:

1 = I was always able to connect to the server

2 = most of the time I was able to connect to the server

3 = most of the time I was not able to connect to the server

11. While Woodstock was running, how often did you lose your connection to the Woodstock Server?

1 = never

2 = most of the time I did not lose the connection to the server

3 = most of the time I lost the connection to the server

### **General**

12. What did you dislike most about Woodstock?

13. What did you like most about Woodstock?

14. Was there any information that you would have liked to see displayed that wasn't displayed?

15. Was there any information that was displayed but that you didn't need at all?

16. What functionality would you like Woodstock to have that it doesn't have now?

17. Did you experience any problems with Woodstock that completely prevented you from using it?

18. Additional comments. (comments are more than welcome!)

# Appendix C: Woodstock Screen Shots



Appendix C: Woodstock Screen Shots

**WoodStock**

My Portfolio | Stock Prices | Trade History | Pending Transactions | Failed Transactions | About

### Portfolio for Jimmy Hendrix

(Stock prices are market closing prices for Wed, Dec 6, 2000)

Cash Available: \$45.01      Total Assets: \$7,510.72  
 Total Stocks Value: \$7,465.71      Loss: (\$2,489.28)

Company Name	Stock Symbol	Stock Price	Number of Shares	Total Value	Profit/ (Loss)	Trade
Apple Computer, Inc.	AAPL	14 5/16	22	\$314.82	(\$114.18)	BUY SELL
EBay, Inc.	EBAY	36 5/16	32	\$1,161.92	(\$696.00)	BUY SELL
IBM	IBM	96 3/4	10	\$967.50	\$20.00	BUY SELL
Intel Corporation	INTC	31 3/4	20	\$635.00	(\$226.20)	BUY SELL
Microsoft Corporation	MSFT	56 11/16	23	\$1,303.87	(\$195.50)	BUY SELL
Oracle Corporation	ORCL	30 3/16	40	\$1,207.60	(\$202.40)	BUY SELL
Yahoo! Inc.	YHOO	37 1/2	50	\$1,875.00	(\$1,075.00)	BUY SELL

WoodStock 1.0 (c) 2000, Brown U

**WoodStock**

My Portfolio | Stock Prices | Trade History | Pending Transactions | Failed Transactions | About

### Stock Prices

(Stock prices are market closing prices for Wed, Dec 6, 2000)

Company Name	Stock Symbol	Stock Price	Open	Low	High	Trade
Amazon.Com, Inc.	AMZN	23 5/8	24 11/16	23 5/8	26	BUY
America Online, Inc.	AOL	44.61	44.49	43.34	45.65	BUY
Apple Computer, Inc.	AAPL	14 5/16	14 5/8	14	15	BUY SELL
Compaq Computer Corp.	CPQ	20.10	22.05	19.50	22.06	BUY
Dell Computer Corporation	DELL	18	19 3/8	17 1/2	19 27/64	BUY
EBay, Inc.	EBAY	36 5/16	38 1/8	35 1/4	39 1/2	BUY SELL
Hewlett-Packard Company	HWP	32	32 1/4	31 1/8	33 13/16	BUY
IBM	IBM	96 3/4	101 3/4	94 13/16	103	BUY SELL
Informix Corporation	IFMX	3 5/16	3 27/64	3 3/16	3 17/32	BUY
Inprise Corporation	INPR	5 9/32	5 1/4	5 1/16	5 15/32	BUY
Intel Corporation	INTC	31 3/4	35 5/16	31 1/4	35 5/16	BUY SELL
Lucent Technologies, Inc.	LU	15 1/8	16 3/4	15 1/16	16 15/16	BUY
Microsoft Corporation	MSFT	56 11/16	60	56 1/16	60 1/2	BUY SELL
Motorola, Inc.	MOT	17 13/16	19	17 5/8	19 1/4	BUY
Oracle Corporation	ORCL	30 3/16	31 3/16	29 5/16	31 5/8	BUY SELL
Rational Software Corp.	RATL	36 7/16	37 5/16	35 1/4	38 7/8	BUY
Red Hat, Inc.	RHAT	5 11/32	6 5/16	5 1/8	6 5/16	BUY
Sun Microsystems, Inc.	SUNW	44 1/4	46 11/16	42 9/16	48 1/8	BUY

WoodStock 1.0 (c) 2000, Brown University / Luis J. Vega      ON-LINE

**Buy Request**

**AOL's Current Stock Price: \$50.50**

If stock price is less than

 \$ 999.99 **buy** 999 shares.

Enter    Cancel

**Sell Request**

**IBM's Current Stock Price: \$93**

If stock price is more than

 \$ 999.99 **sell** 999 shares.

Enter    Cancel

Appendix C: Woodstock Screen Shots

WoodStock

My Portfolio | Stock Prices | Trade History | Pending Transactions | Failed Transactions | About

Date transaction was requested	Transaction Type	Stock Symbol	Shares Requested	Price per share requested	Price per share obtained	Transaction dollar amount
11/10/00 @ 10:52:07 PM	Buy	AAPL	2	Less than \$999.99	19 1/2	\$39.00
11/10/00 @ 10:52:14 PM	Buy	INTC	20	Less than \$999.99	43 1/16	\$861.20
11/10/00 @ 10:52:18 PM	Buy	ORCL	40	Less than \$999.99	35 1/4	\$1,410.00
11/10/00 @ 10:52:23 PM	Buy	AAPL	20	Less than \$999.99	19 1/2	\$390.00
11/10/00 @ 10:52:33 PM	Buy	YHOO	50	Less than \$999.99	59	\$2,950.00
11/10/00 @ 10:52:40 PM	Buy	EBAY	32	Less than \$999.99	58 1/16	\$1,857.92
11/10/00 @ 10:52:48 PM	Buy	MSFT	23	Less than \$999.99	65 3/16	\$1,499.37
11/10/00 @ 10:53:06 PM	Buy	IBM	10	Less than \$999.99	94 3/4	\$947.50

WoodStock 1.0 (c) 2000

WoodStock

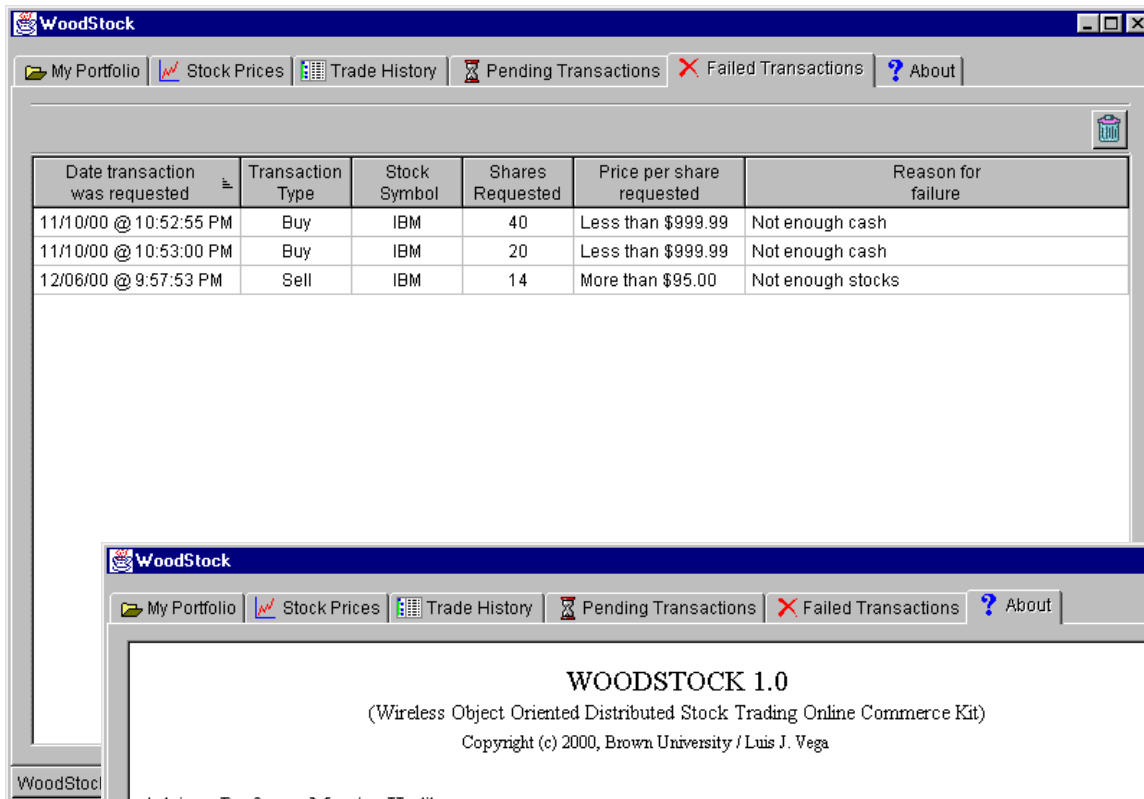
My Portfolio | Stock Prices | Trade History | Pending Transactions | Failed Transactions | About

Date transaction was requested	Transaction Type	Stock Symbol	Shares Requested	Price per share requested	Sent to Server?
12/06/00 @ 10:04:41 PM	Sell	AAPL	15	More than \$10.00	No
12/06/00 @ 10:04:58 PM	Buy	INTC	57	Less than \$35.00	No

WoodStock 1.0 (c) 2000, Brown University / Luis J. Vega

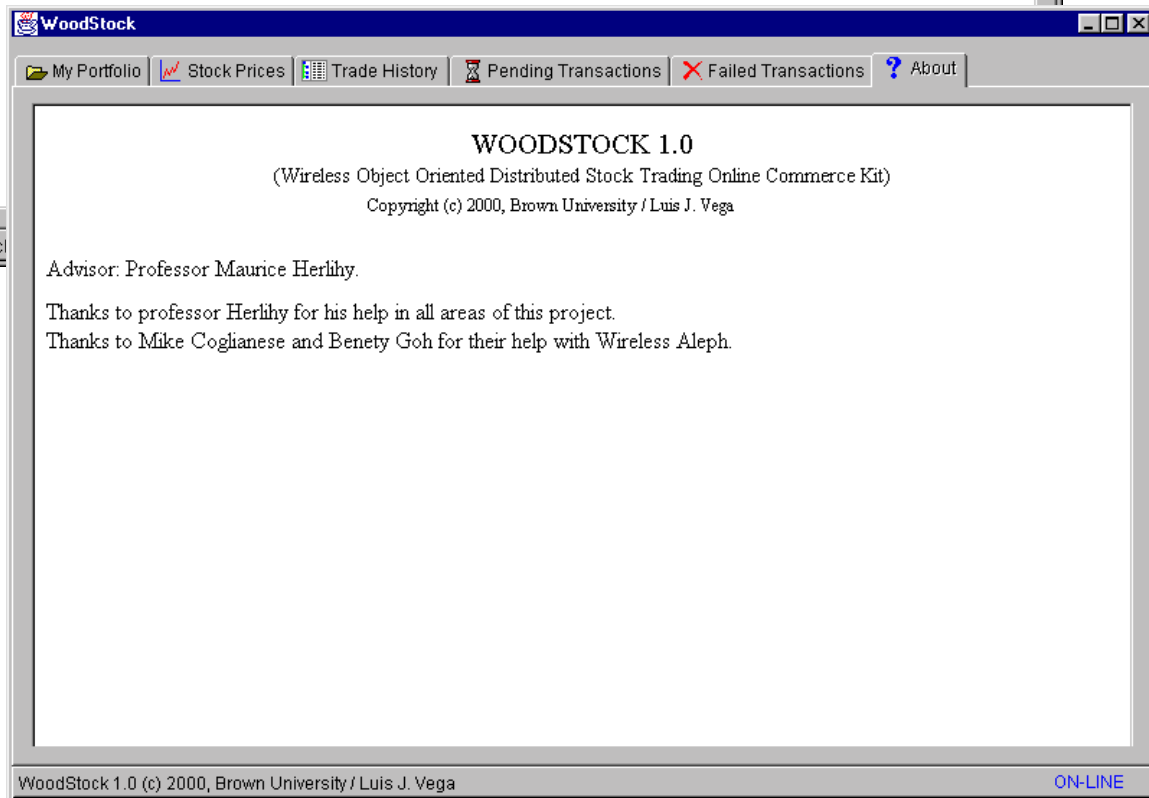
OFF-LINE

Appendix C: Woodstock Screen Shots



The screenshot shows the WoodStock application window with a menu bar containing 'My Portfolio', 'Stock Prices', 'Trade History', 'Pending Transactions', 'Failed Transactions', and 'About'. The main content area displays a table with the following data:

Date transaction was requested	Transaction Type	Stock Symbol	Shares Requested	Price per share requested	Reason for failure
11/10/00 @ 10:52:55 PM	Buy	IBM	40	Less than \$999.99	Not enough cash
11/10/00 @ 10:53:00 PM	Buy	IBM	20	Less than \$999.99	Not enough cash
12/06/00 @ 9:57:53 PM	Sell	IBM	14	More than \$95.00	Not enough stocks



The screenshot shows the WoodStock application window with the 'About' dialog box open. The dialog box contains the following text:

**WOODSTOCK 1.0**  
(Wireless Object Oriented Distributed Stock Trading Online Commerce Kit)  
Copyright (c) 2000, Brown University / Luis J. Vega

Advisor: Professor Maurice Herlihy.

Thanks to professor Herlihy for his help in all areas of this project.  
Thanks to Mike Coglianese and Benety Goh for their help with Wireless Aleph.

WoodStock 1.0 (c) 2000, Brown University / Luis J. Vega ON-LINE

# Appendix D: Woodstock Web Site

Woodstock - Microsoft Internet Explorer provided by EarthLink Network, Inc.

File Edit View Favorites Tools Help

## Welcome to WOODSTOCK !

WoodStock

My Portfolio Stock Prices Trade History Pending Transactions Failed Transactions About

**Portfolio for Jimmy Hendrix**  
(Stock prices are market closing prices for Fri, Nov 10, 2000)

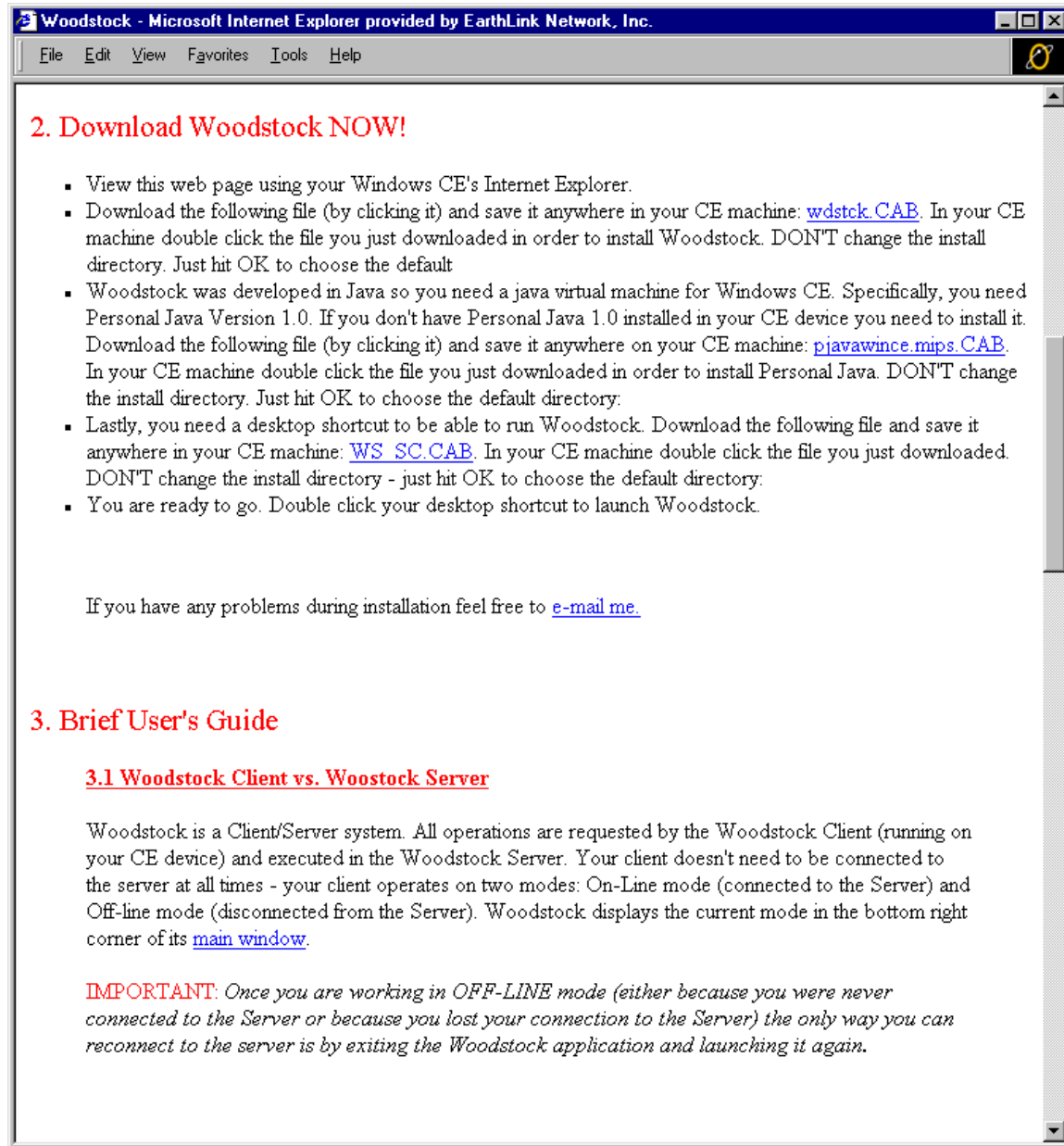
Cash Available: \$45.01 Total Assets: \$8,997.59  
Total Stocks Value: \$8,952.58 Loss: (\$1,002.41)

Company Name	Stock Symbol	Stock Price	Number of Shares	Total Value	Profit/ (Loss)	Trade
Apple Computer, Inc.	AAPL	19 1/16	22	\$419.32	(\$9.68)	BUY SELL
EBay, Inc.	EBAY	46 1/16	32	\$1,473.92	(\$384.00)	BUY SELL
IBM	IBM	93	10	\$930.00	(\$17.50)	BUY SELL
Intel Corporation	INTC	37	20	\$740.00	(\$121.20)	BUY SELL
Microsoft Corporation	MSFT	67 3/8	23	\$1,549.74	\$50.37	BUY SELL
Oracle Corporation	ORCL	25 7/16	40	\$1,017.60	(\$392.40)	BUY SELL
Yahoo! Inc.	YHOO	56 7/16	50	\$2,822.00	(\$128.00)	BUY SELL

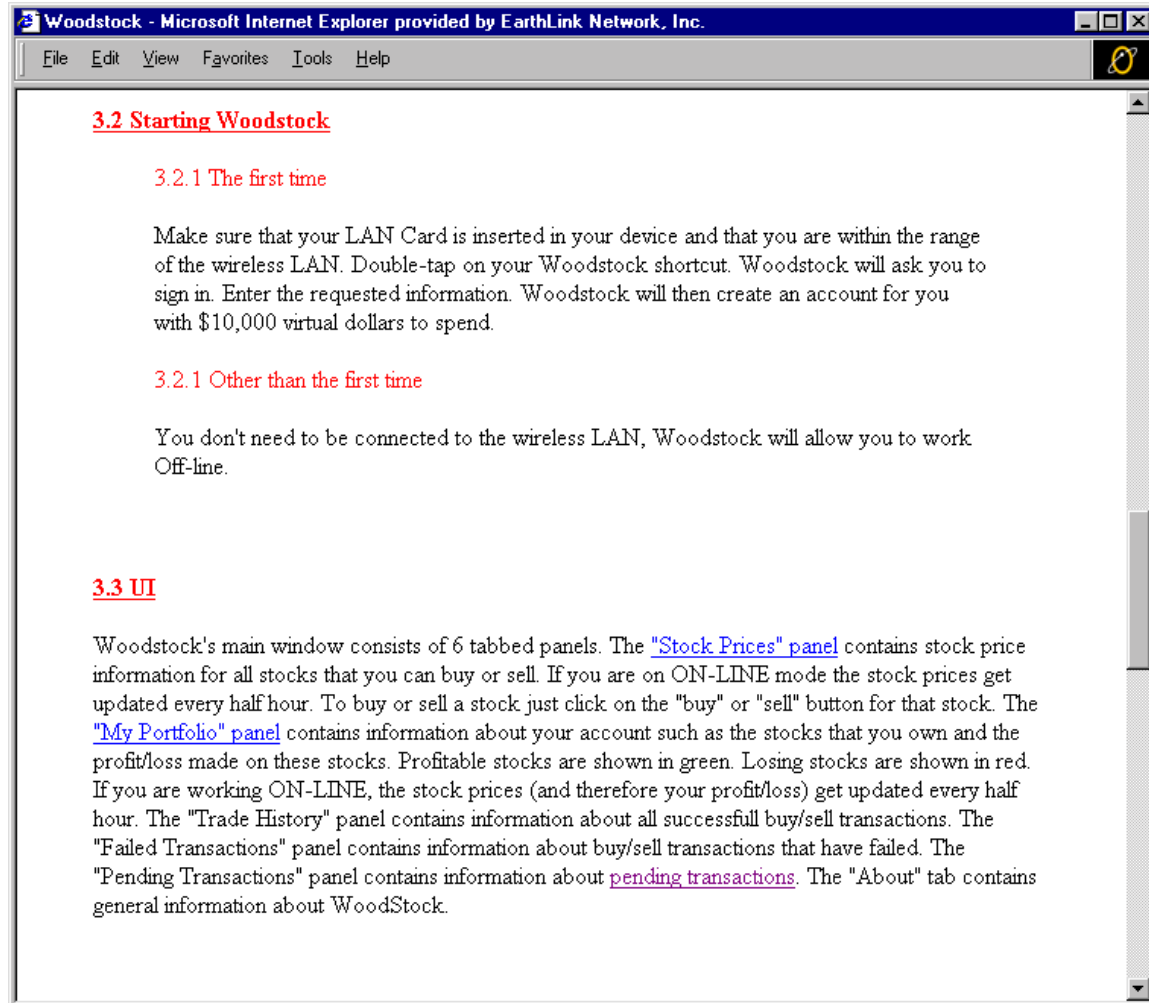
WoodStock 1.0 (c) 2000, Brown University / Luis J. Vega OFF-LINE

### 1. What is Woodstock?

Woodstock is a stocks trading game developed for the Windows CE operating system. Players are given \$10,000 "virtual" dollars which they can use to buy stocks. The stock prices presented to the user are the actual prices of the NYSE and NASDAQ markets (although they are only updated every 30 minutes). When the game is over, the user with the largest profit is the winner.








**3.4 Buy/Sell Transactions**


To buy/sell stocks you tap the "buy" or "sell" button from the ["My Portfolio" panel](#) or the ["Stock Prices" panel](#).

**3.4.1 Buy Request**



You must enter the number of shares you want to buy and a price below which you are willing to buy these shares.

**3.4.2 Sell Request**



You must enter the number of shares you want to sell and a price above which you are willing to sell these shares.

**3.4.3 Pending Transactions**

A transaction is considered pending if (1) you requested a buy/sell transaction but it has not yet been sent to the Woodstock Server (most likely because you are working in OFF-LINE mode) or (2) you requested a buy/sell transaction and it was sent to the Woodstock Server but your Woodstock client has not yet received a confirmation about this transaction from the Server.

A pending transaction that has not been sent to the Woodstock server because you are working in OFF-LINE mode will be sent to the server the next time a connection to the server is established. Note that *the only way you can reestablish a connection to the server is by exiting the Woodstock application and launching it again.*