CocoaFIBS

A Mac OS 10 Client for the FIBS Backgammon Server

CS Master's Thesis

Adam Gerson

Pace University

December 29, 2004

http://homepage.mac.com/agerson/cocoafibs/

Outline

1. Introduction

The goal of this project is to design, release, and maintain a freeware client application for the First Internet Backgammon Server(FIBS) that runs natively in Mac OS X. This project involves the exploration of the FIBS Client Protocol, the Objective-C language, the Cocoa Frameworks, Mac OS X development tools, user interface and graphic design, coordinating alpha and beta testing, socket programming, and the Model-View-Controller paradigm.

FIBS is a backgammon server that has been around for twelve years and has supported Macintosh users since the day it opened its doors. It has been nearly four years since the release of Mac OS 10.0 and no native client exists to connect to the FIBS server from a Macintosh. All Mac users connect with a Java Applet, Java Application or emulated Mac OS 9 client. This tool will serve as a valuable contribution to the FIBS community.

Mac OS 10 uses Objective-C as its main development language and a series of GUI frameworks called Cocoa. Objective-C is an object oriented ANSI C superset. The free development environment that ships with Mac OS X called XCode and the Objective-C / Cocoa combination offer a developer a rich experience. The secondary goal of this project was to learn and explorer these tools.

The design of this tool will leverage the power of the Model-View-Controller paradigm and the Apple Aqua user interface controls. Aqua is Apple's code-name for Mac OS 10's GUI elements. This tool consists of over 14 custom classes and 2 classes written by others. They are a combination of sockets, GUI controllers, model objects, and custom drawn views. Attempts were made at every turn to leverage the beauty and power of the Mac.

The core philosophy of this application is minimalist. FIBS Clients have the potential to be bloated. This combined with Mac OS 10's precedent for aesthetics are the impotence for a minimalist design. With the exception of the login window, the application is contained within a single window. The game board, a animated user list, tool bar, and three chat and console views are all elegantly contained within a single customizable window. The user list can be hidden and the toolbar can be customized. This allows the user to save space and have only the necessary buttons in front of them at any one time. The FIBS server is a telnet server. This application interprets user interaction and sends commands to the server. In return all commands sent by the server are shown in the GUI.

The prime challenge of writing this application was a complete lack of experience developing Macintosh applications. To develop this application I had to learn Objective-C, Cocoa and XCode . The messaging syntax of Objective-C is maddening to someone who has grown up on Java. For instance, here is an example of an Objective-C method call and a similar call in Java syntax:

[userListWindowData objectForKey:@"client"];	**Objective-C**
userListWindowData.objectForKey("client");	**Java**

An example of Objective-C Vs. Java messaging syntax.

The next challenge was to understand the FIBS Client Protocol. My experience with the server has only been as the user of a GUI client application. Both The FIBS Client Protocol and Objective-C are well documented on the web.

The next step in the project is to compare my client with existing clients. My client had the virtue of running natively, but many of these client have been in development for years and had some impressive features. The strongest plus of my application is its simple single-window design.

The final challenge is to release and test and re-release the client into the actual community. I have found a great many willing users as alpha testers and people willing to donate their graphic design time.

2. Background

When I was a senior in high school my pregnant math teacher, Erika Batiuo, was forced to leave in the middle of the year because of unforeseen complications with her pregnancy. My school was apparently unable to find a suitable replacement. For the remainder of the year, which amounted to almost an entire semester, our math class was taught by the biology teacher. Robert Costello, though a wise man and passionate scientist, did not have the ready knowledge to carry on our calculus curriculum. He decided to share with us his one pseudo-mathematical passion: Backgammon. He first taught us the rules of the game, then we moved on to basic strategy, probability, and ended the year with a ladder tournament from which I was eliminated in the second to last round.

After graduating from high school, my friends and I continued to play, read about, and learn the game the Turkish call “Shesh-Besh” which means 6-5 in English. After getting to college in 1997 I taught some friends and discovered Yahoo Games, a site where people can play multi-player board games online. However, the browser-based Java interface of Yahoo Backgammon was slow and clunky. Discovering the FIBS server and the active user-community that participated in it was a very exciting event for me. I had a native Mac OS 9 GUI client running on my laptop and endless chat rooms of mentors eager to discus strategy, food, and politics. I have been playing on FIBS ever since.

2.1 The Game

Backgammon is a game of both skill and luck. The luck factor is derived from the use of two, six-sided dice. Games involving dice appear to have developed independently in all cultures of the world. The ancestors of Backgammon date back almost five thousand years to ancient Mesopotamia in what is present day Iraq. Backgammon's history appears to indicate that it is the oldest known board game in Human history. The modern version of the game appeared sometime in England during the 17th Century. The goal of the game is to use strategy and the luck of the dice to move all of your pieces around, and off, the board before you opponent.[1]

2.2 FIBS

People have been playing backgammon on computers since the earliest days of computer gaming. The first computer that could play Backgammon at the "Expert Level" used neural networks and was named TD-Gammon. This program was developed by Dr. Gerald Tesauro of IBM. [2] The appropriately named First Internet Backgammon Server(FIBS), was the first online backgammon community and remains free to this day. FIBS was created by Andreas Schneider and opened its doors on July 19, 1992. The first members of FIBS used a simple telnet client to connect, play and chat. In 1994, Windows and Macintosh graphical clients were released by independent members of the FIBS community. All graphical FIBS clients are essentially interpreting and sending telnet commands in response to user actions. The ongoing maintenance of the server was abandoned by its creator, but is now maintained and developed by volunteers. The source code for the server is not open source. [3]

2.2.1 FIBS Research & Literature

To write a graphical FIBS client one must have a robust understanding of the telnet commands and response types used by the server. The FIBS Command Reference is intended for telnet end-users and provides a detailed description of all commands understood by the FIBS server. The FIBS Client Protocol Detailed Specification is a reference intended for programmers who are writing graphical interfaces to the FIBS server. This document not only provides a detailed description of the commands, but also includes breakdowns and examples of all server responses to commands and their arguments. The formal specification is at version 1.008 and was last updated in 1997. The specifications for the FIBS CLIent Protocol (CLIP) differ slightly from the end-user FIBS Command Reference. Clients can log into the telnet server with a special syntax that returns CLIP formatted responses for parsing as opposed to ones intended for a human to view with their eyes.

My research for understanding the FIBS Client Protocol consisted of reading these documents in detail, playing telnet games logged in as both a user and as a "client", and downloading the various other graphical clients available for other operating systems.

2.3 The Cocoa Framework and Objective-C Language

Objective-C is a object oriented superset of ANSI C. Its infix syntax is similar to Smalltalk-80 and nothing like C++ or Java. Objective-C was written in the 1980's by

Brad Cox while at the StepStone corporation. In 1988 it was adopted by the virgining NeXTstep platform as its development environment. It was made a part of the GNU gcc compiler in 1992. Objective-C's future was not looking extremely bright until two major events took place. These events were Steve Jobs return to Apple Computer and NEXT's sale to Apple. Sometime in the late 1990's it was decided that the next major evolution of Apple's Macintosh Operating System would be based on NeXTstep. Thus, Objective-C is now used as the primary development language for MacOS X applications, though most of the Mac OS X Operating System itself is written in C. Objective-C is also the language for the GNUstep project on Linux. Objective-C is distinguished from C++ and Java by its weak typing and runtime features. [4]

Cocoa is a set of object-oriented frameworks for the development of Mac OS X-only native applications. Cocoa is also used to build the stunning Aqua user interfaces expected by Mac users. Cocoa also draws its history from NeXTStep. Cocoa is implemented in Objective-C. However Cocoa frameworks can be used by a variety of different languages. "Cocoa applications can make use of core functionality contained in traditional C and C++ libraries brought forward from legacy application environments. It also provides Java interfaces to permit a high performance, full featured Java development environment tailored to Mac OS X specifically."[5] This robustness was extremely useful during the development of CocoaFIBS.

2.3.1 Cocoa / Objective-C Research & Liturature

Learning development on MacOS X was a multi-stepped process. This process consisted of reading three books: Cocoa Programming (Anguish, 2003), Cocoa Programming For Mac OS X (Hillegass, 2004), and Learning Cocoa with Objective-C (Davidson, 2002). These books took me through simple tutorials, example projects, and general philosophy and served as a foundation for my trip to The Big Nerd Ranch.

> "Gone are the days of dull…training. Big Nerd Ranch offers intensive, head-down computer programming courses taught by experts in a retreat environment. Classroom, accommodations, and dining all take place within the same building, freeing you to learn, code, and discuss with your programming peers and instructors. At Big Nerd Ranch, we take care of the details; your only job is to learn."[6]

Spending five days at The Big Nerd Ranch in the backwoods on Atlanta, GA with Aaron Hillegass was an overwhelming experience of learning, natural seclusion and a torrent of information.

On the first day of the class Aaron pronounces, "There's a lot to learn. Today will be like drinking from an open fire hydrant. It will be messy and your face is going to get wet. You'll probably feel lost and exhausted and you may even hate me by the end of the day. But the next day will be better and, by the end of the week, you'll be building Cocoa apps effortlessly." [7] This week of intense training and discussion prepared me for my first major Macintosh project: The development of my FIBS client which I will call CocoaFIBS.

2.4 Other Tools

There are many other clients for the FIBS server. FIBS/W, the first Windows client was released in 1994 by Robin Davies and is no longer maintained. The first Macintosh client, also released in 1994, was MacFIBS by Paul D. Ferguson. MacFIBS was originally written for MacOS 7, and updates were maintained through MacOS 8 and 9. Paul still supports MacFIBS, but has no plans to release an OS X compatible version. He now works for a company that develops commercial backgammon clients for a pay server. Over the years clients for almost ever major operating system have been developed. Windows user have a choice of three different clients, 3DFiBs, CFFIBS, and RealFibs. Linux user can choose from bacKgammon or KBackgammon. Cell phones and palmtops can run MobileFIBS and FibsCE. JavaFIBS is available as both a Browser based client and as an executable .jar file. User's of any OS with a telnet client can, of course, always connect the old fashioned way. No native Mac OS X client exists for the FIBS server. Mac OS X users represent a fraction of the FIBS community and no one has yet committed the time and skill required to develop a Mac client.

3. CocoaFIBS

3.1 FIBS CLIP Protocol

The FIBS CLIP Protocol is an attempt to make client-server communication more understandable for interfacing programs. It requires a special login syntax that tells the server it is going to communicate with a program rather then a human being. A CLIP message consists of a single line of ASCII text terminated by carriage return / line Feed. CLIP messages are the most basic form of client communications with the FIBS server.

> "The CLIP or FIBS CLIent Protocol is an effort to make the FIBS server a little more friendly for client programs to interface with. FIBS was originally designed for use with a straight telnet session with a command line like interface. When you login in CLIP mode the main differences from the telnet mode [Standard Log In] are: There is no ">" prompt. You get the various CLIP messages documented…instead of their logical counterparts in telnet mode. All the clip commands start with a number. Most of them will occur asynchronously and not always as a direct result of sending a command."[8]

The commands sent by the server in CLIP mode are prefixed with an unique id code and are easily parsed by any program capable of reading and interpreting character delimited strings. The CLIP expects the behavior layout in the formal telnet protocol specification(RFC 854). The author of the CLIP Specification summarizes the relevant information.

> "The most important aspect of this is that FIBS insists that a new line is always indicated by sending (or receiving) both a carriage return (CR=13) and a line feed character (LF=10). Following the conventions of the C programming language I'll refer to CR as \r and to LF as \n. FIBS also tries to be a good RFC 854 Network Virtual Terminal (NVT), ie it tries to be as dumb as possible. It won't send back characters it receives and it won't do any line editing (don't send a backspace character and expect FIBS to handle it). The telnet protocol allows clients and servers to start with the assumption that the server is a dumb NVT and negotiate advanced terminal capabilities. Those are sent as three byte sequences IAC XX YY, where IAC is always the byte 255 (hex $FF) and XX and

> YY are bytes describing the option being negotiated. You might encounter a few of those during the login procedure when FIBS tries to convince the telnet client not to show the password on the screen. Just ignore any 3 byte sequence starting with IAC. Apart from the login sequence FIBS will never send any other 3 byte telnet sequences unless you provoke it and send one yourself (in which case the answer sequence, if any, means 'leave me alone')." [9]

This excerpt for the from the CLIP Specification shows a typical example of a CLIP message description. The number 1 represents the line prefix number that corresponds to the "CLIP Welcome". The three parameters it returns are: Name – Your login name, Last Login – The last time you logged in. The date is measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT), Last Host – The IP address or hostname associated with the last login.

CLIP Welcome

Synopsis:

```
1 name last_login last_host
```

Example:

```
1 myself 1041253132 192.168.1.308
```

This is the very first line you'll see after a successful standard login in client mode. This is the only time you will see this message.

name:	The login name you just logged in as.
last_login:	The time of the last login given in seconds since 00:00:00 UTC on January 1, 1970.
last_host:	The host or IP address the last login came from.

3.2 CODE STRUCTURE

The code used in CF adopts the Model-View-Controller design pattern. The MVC paradigm looks at the big picture when designing an application. A well designed MVC programmer will define all objects as either model objects, view objects, or controller

objects. This provides a clear separation between objects and makes objects more reusable and flexible.

Model objects represent data. They contain a complete set of relevant data for the object they are modeling as well as getter and setter methods to manipulate that data. Ideally a model object is completely encapsulated and knows nothing about the view object. This enables model objects to be re-used with any number of different views.

View objects display information to the user. Often they are displaying data from a model object. In fact, they should contain specific knowledge on how to get and edit data from the model objects. View objects are less reusable and contain more custom code written specifically for a given model.

Controller objects are the glue that bring model and view objects together. Actions that take place in the view are handled by methods in the controller which in turn updates or gets some data from the model. Controller objects are rarely reusable and are often application specific.

3.3 Graphical User Interface Structure & Design

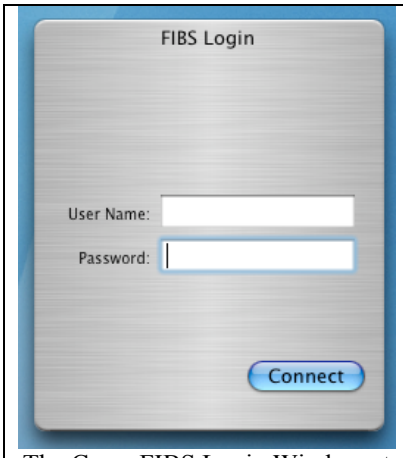
The CF Graphical User Interface was designed in accordance with The Apple Human Interface Guidelines. This document outlines the specific guidelines for designing a user interface in compliance with Mac OS X standards for esthetics and functionality. The overarching principals and goals of The Apple Human Interface Guidelines are:

- Users will learn your application faster if the interface looks and behaves like applications they're already familiar with.
- Users can accomplish their tasks quickly, because well-designed applications don't get in the user's way.
- Users with special needs will find your product more accessible.
- Your application will have the same modern, elegant appearance as other Mac OS X applications.
- Your application will be easier to document, because an intuitive interface and standard behaviors don't require as much explanation.
- Customer support calls will be reduced (for the reasons cited above).
- Your application will be easier to localize, because Apple has worked through many localization issues in the Aqua design process.
- Media reviews of your product will be more positive; reviewers easily target software that doesn't look or behave the way "true" Macintosh applications do.[10]

This concept is reinforced by Joel Spolsky in his book User Interface Design For Programmers, where he succinctly states "A user interface is well-designed when the program behaves exactly how the user thought it would."[11]

3.3.1 The Login Window

The goal of the login window is to be simple, elegant, modern, and clean. No instructions are necessary. All feedback is intended to be visual. The user enters their username and password and clicks the "Connect" button. During the login processes the button and the two text fields disable preventing user interaction. An animated progress indicator spins until login is complete. If there is an error during login a message is displayed to the user, the controls enable, and the user may try and log in again.



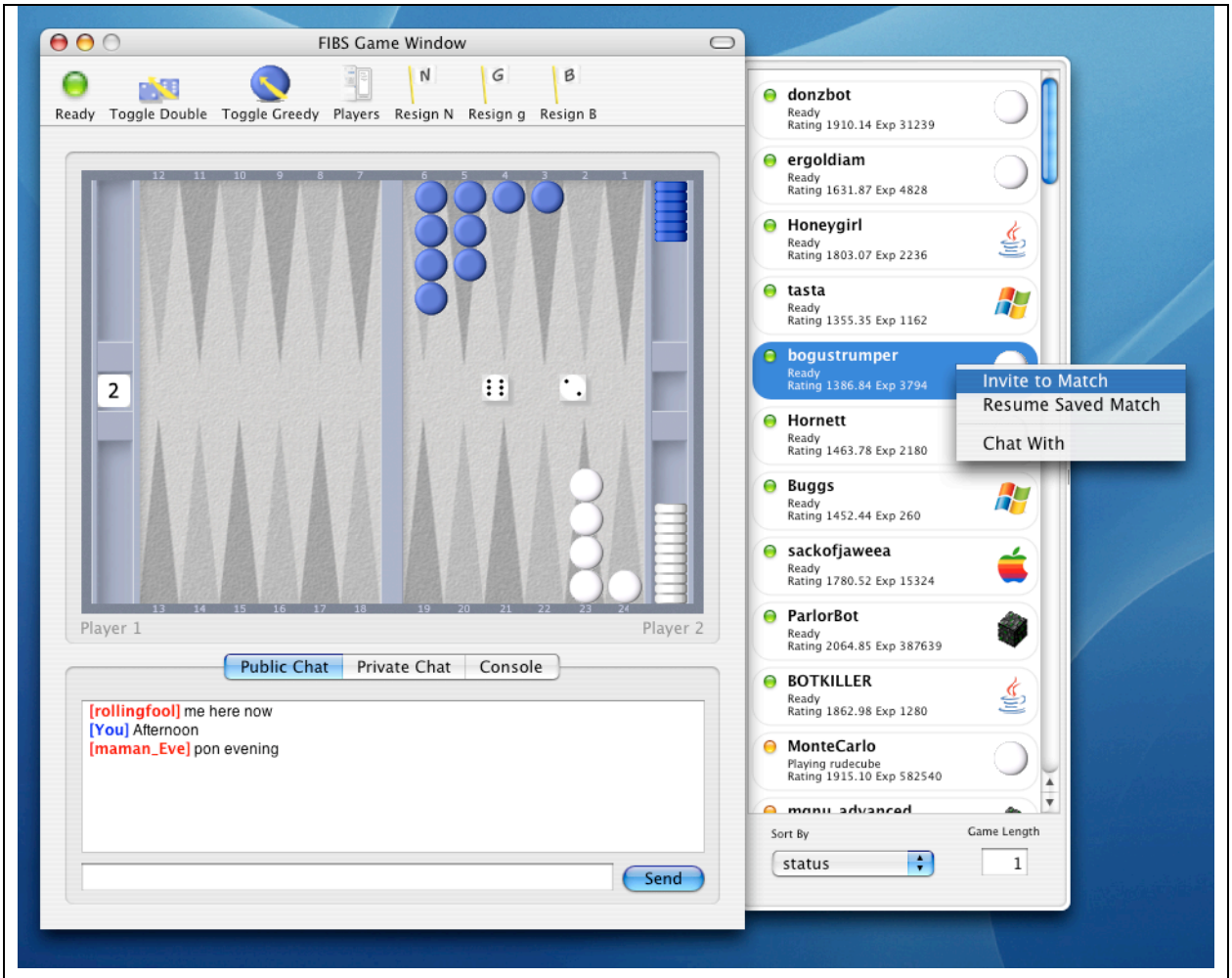
The CocoaFIBS Login Window at application launch time.

The CocoaFIBS Login Window after the user clicks "Connect"

The CocoaFIBS Login Window after an error.

3.3.2 The Game Window

After a successful login the login window automatically hides and the game window shows. This is the main window of the application and most clearly demonstrated the all-in-one unified design of CocoaFIBS. The four major types of information and interaction, toolbar, game play, user information, and chat, are displayed in modular sections on this window.



A screen shot from the CocoaFIBS Main Window

3.3.3 The User List Drawer

The list of connected users and their attributes is displayed in a Mac OS X window called a drawer. "A drawer is a special type of window that can slide out from [either] side of a window."[12] The drawer is toggled by a button in the toolbar or by a menu item in the "View" menu. Inside of the drawer is a field for setting the desired game length. This value is sent to a prospective player when inviting them to play a game. There is also a pop up button for sorting the list of users.

Some cells from the table of users

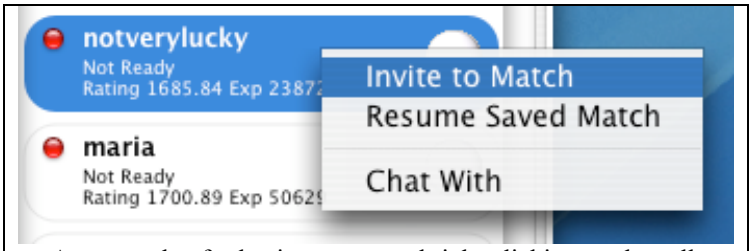
The list of users is stored in a table with custom drawn cells. Each cell contains information about an individual user. The colored "bubble" in the top left hand corner of each cell represents status.

Green = ready to play
Yellow = playing a game
Red = not ready to play

The cells contain the name of the player, more detail on their status, their rating, experience, and an icon representing their operating system.

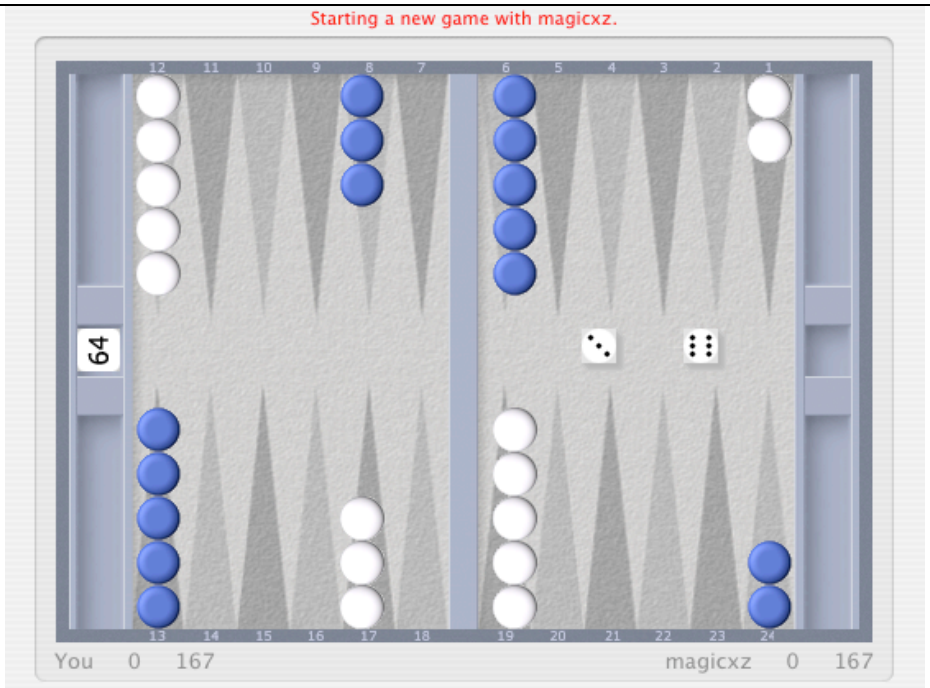
A single cell from the table of users

When the player right-clicks on a cell they get a pop up menu that offers them the choice to invite the user to a match, resume a saved match, or chat with the user.



An example of selecting a user and right-clicking on the cell.

3.3.4 The Game Window



The game window at the start of a new match.

To start a new match you may invite a user or accept the invitation of a user. The game board consists on the chips which during your turn can be dragged to and from triangles. Clicking on the dice ends your turn and clicking on the doubling cube offers a double to your opponent. The two players names, the score of the game, and the pip count are kept at the bottom of the window. System and error messages regarding the game are displayed in red at the top of the window. In-game choices appear in the form of an animated sheet that slides down from the top of the window.

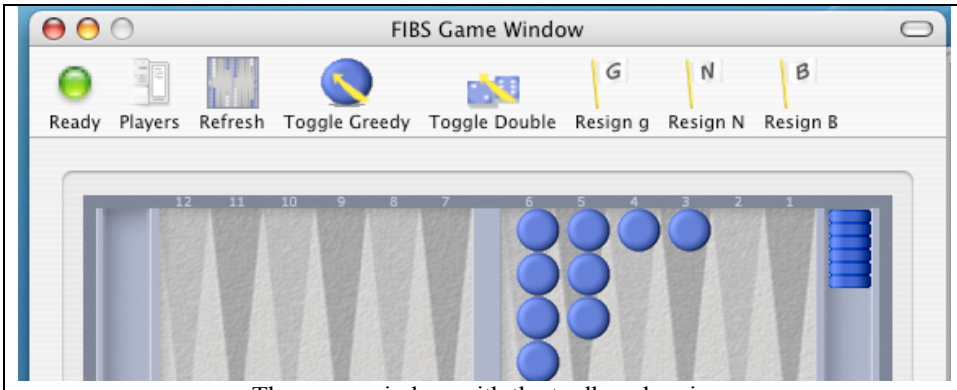
A sheet in the game window request feedback from the player.

3.3.5 The Toolbar

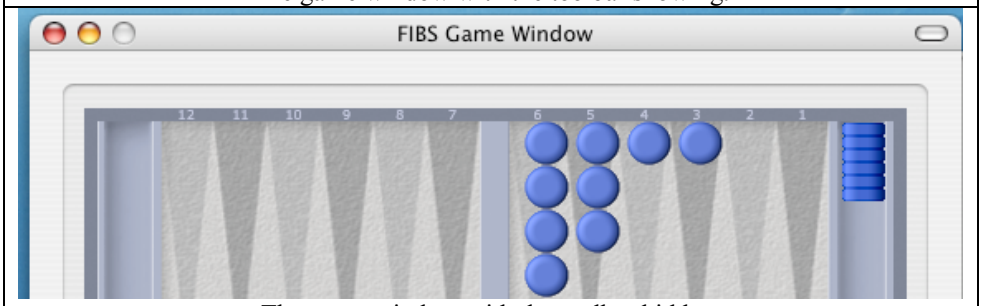
The toolbar at the top of the game window enables quick and customizable access to some common game options.

Ready – Toggles the players ready to play status from ready to not ready.
Players – Hides and shows the user list drawer.
Refresh – Get a new board from the server and acts as an undo button for moves.
Toggle Greedy – Toggles the use of "greedy play" a feature where the server moves for you.
Toggle Double - Toggles the use of auto-rolling
Resign n (normal) –You'll lose the number of points on the cube.
Resign g (gammon) - You'll lose twice the number of points on the cube.
Resign b (backgammon) - You'll lose three times the number of points on the cube.

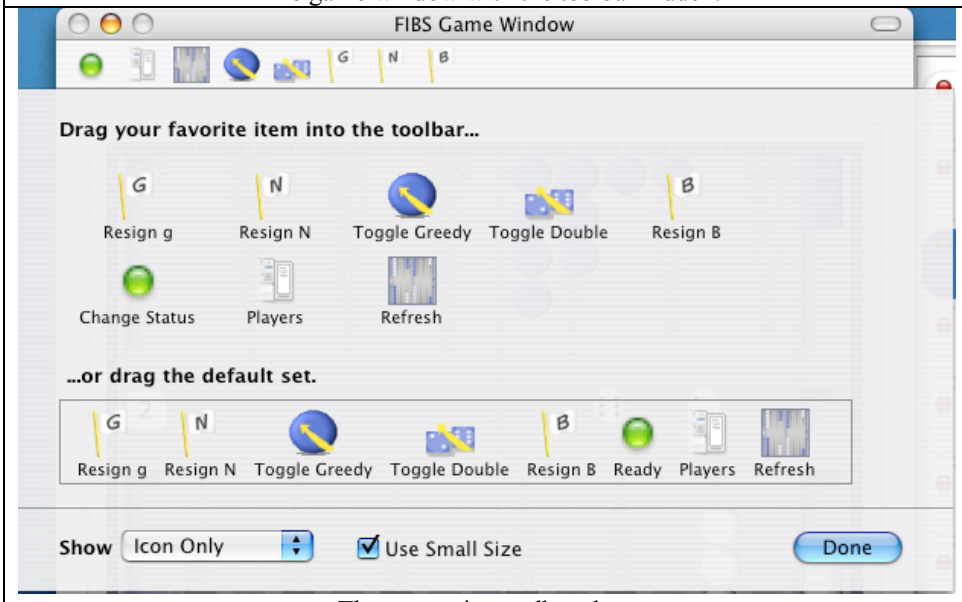
The location and presence of any toolbar items is customizable. The toolbar can also be hidden to save space. Other view options include the ability to use smaller icons, icons only, or text only.



The game window with the toolbar showing.



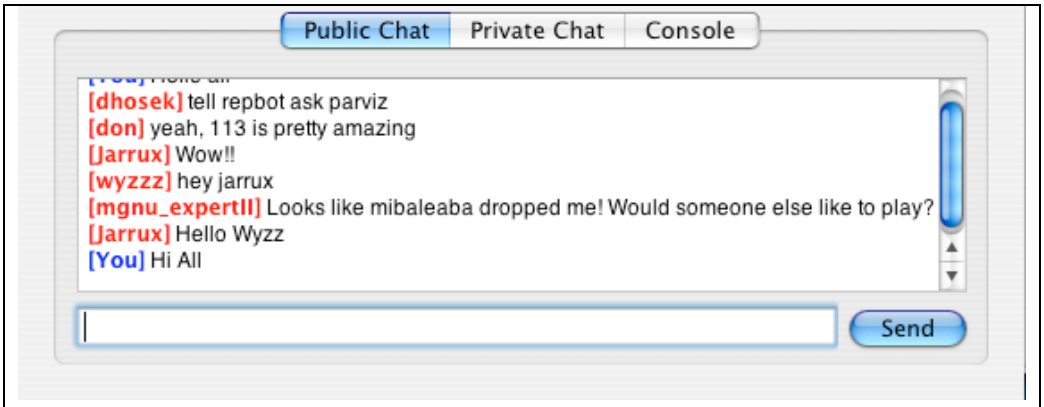
The game window with the toolbar hidden.



The customize toolbar sheet.

3.3.6 The Chat Tab Panel

The chat tab panel located at the bottom of the screen encapsulates the 4 major types of interactive communication. Public chat forums, private one-on-one chat, in game chat, and console communication can all be access from one small area.



The chat tab panel show an example of "public chat".

4. Comparisons

4.1 JavaFIBS Application

JavaFIBS is the quintessential Java application. Using it gives one the feeling that they have left their own operating system universe only to enter a bizarre parallel dimension. Sun's Java GUI, which was designed by the same people who brought us The OpenWindows project, is ugly and confusing. The first major problem with using a Java Application is that the GUI is different from the one you are used to. Well written operating systems are designed so that as the user learns the "metaphors" and standards of the OS they become more proficient and productive. As stated in the Apple Human Interface Guidelines "Users will learn your application faster if the interface looks and behaves like applications they're already familiar with." Launching JavaFIBS breaks this connection and forces the user to not only learn a new Application, but to learn a new OS as well.

The second problem with the Java GUI is that it is ugly. The widgets that comprise the GUI such as buttons, scroll bars, lists, and dialogue boxes, look like they belong in 1989. They lack the modern look and feel that users have come to expect. In fact, they resemble the GUI of the X-Window system written in 1984. I do not intend to show any disrespect to Java. The concept of portable code being able to run anywhere is marvelous. However, it is just not there yet in its ability to compete with applications written natively for the hardware they were intended to run on.

Thus, JavaFIBS starts it endeavor with a handicap. Its written in Java. CocoaFIBS attempts to include as much content as possible within a single window. JavaFIBS shares this characteristic. However, JavaFIBS has so much content whizzing past the user at any time that it is hard to focus on any single element. In fact, the main window takes up too much space on the screen. JavaFIBS also presents a lot of its feedback in text form, as opposed to CocoaFIBS which uses a combination of text, coloring, and icons. "…users hate reading, and will avoid it unless they absolutely cannot accomplish their task."[13]

CocoaFIBS offers almost every screen that JavaFIBS does in half the space. Granted JavaFIBS does allow the user to customize the size of every subsection of every window. However, most novice user interface users don't even understand this. Yet, even with customization it is hard to cajole JavaFIBS into a visual pleasing and efficient layout. The JavaFIBS window is divided into seven sections. The players/board screen, the private chat screen, the public chat screen, the system/friends/villains screen, the invitation screen, the toolbar and the menus.

The players/board screen works well, but initially provides too much information. This screen presents the user with a tab that lets them switch between the board and a list of users. Since the only purpose of the user list is too invite people to play a game then there is no reason why one would need access to these two screens at once. However, the idea of a backgammon application without a backgammon board showing at all time leaves a blandness to the visual experience of the application. CocoaFIBS ever present backgammon board is pleasant and the problem of the user list taking u p space is solved

by the hideable drawer that CocoaFIBS employs. The user list in JavaFIBS presents way too much information for an initial screen. Column headers such as gender, country, email and interesting, but belong in a “more info” screen somewhere, and are not necessary for choosing an opponent. The user list in CocoaFIBS is exponentially smaller and provides only the essential information such as name, status, ranking, experience, and operating system. Both programs allow the user to sort the list by a variety of criteria.

In general the chat section takes up way to much space in an application window that is already to big. That chat font is not anti-aliased and the colors are drab not indicate the status of chat as easily is in CocoaFIBS which does use font anti-aliasing.

Another section is devoted to a tab-panel that lets the user switch between system output, a friends list and a villains list. Aside from the fact that these three screens have nothing to do with each other, they are not a very prevalent part of the game. In fact, most user’s have no interest in system messages at all. Yet, this screen has the same central prominence as the game board itself. In CocoaFIBS system messages are available in two intuitive way. Important system messages are displayed in red above the game board. If the user would like to view all commands and messages returned from the server there is a separate tab in the chat section of the main window. Though the ability to mark users as friends or enemies is ingenious, devoting a separate space from them is not. The game already has a list of players. Simple color coding or icons in the existing user list could easily denote this status.

Java FIBS devotes an entire section to a list of all invitations received, CocoaFIBS uses a dismissible dialogue box. One a user receives an invitation to play there are only two decisions; accept the invitation or decline the invitation. After that there is no need to keep the information around when screen real estate is a factor.

Both CocoaFIBS and JavaFIBS employee a toolbar of buttons at the top of the screen. However, CocoaFIBS toolbar is customizable and hidable while JavaFIBS offers neither of these options. The icons JavaFIBS uses are non-descriptive and confusing, Waiting for the mouseOver tooltips, which CocoaFIBS also has, is the only way to know what the buttons do.

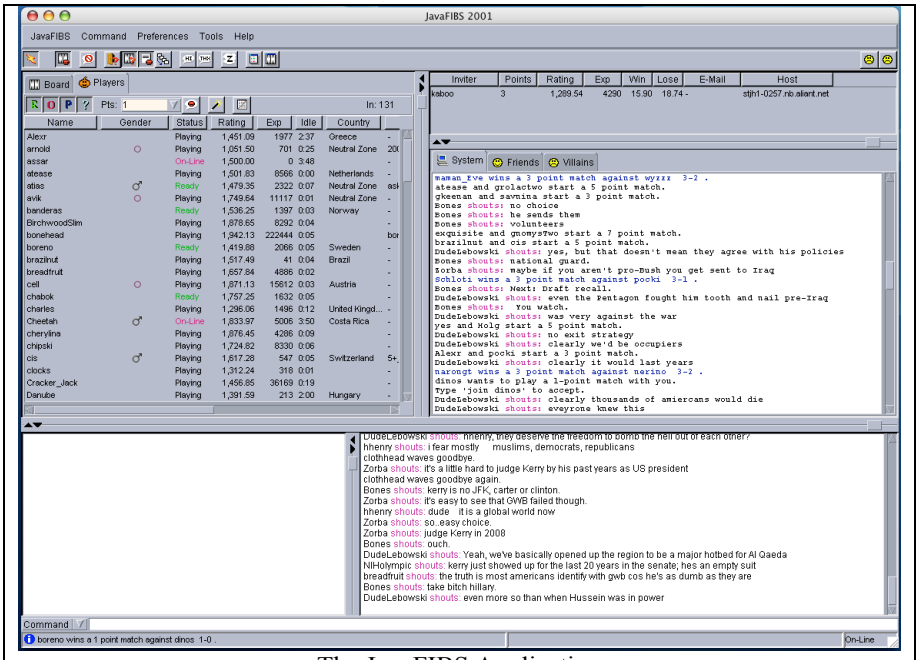
	A JavaFIBS button that toggles the showing and hiding of commands sent to the server.

Another problem with JavaFIBS, or any Java application for that matter, is the "in application" menu bar.

> "When you want to point to the File menu…you have a target about half an inch wide and a quarter of an inch high to acquire. You must move and position the mouse fairly precisely in both the vertical and the horizontal dimensions. But on a Macintosh, you can slam the mouse up to the top of the screen, without regard to how high you slam it, and it will stop at the physical edge of the screen - the correct vertical position for using the menu. So, effectively, you have a target that is still half an inch wide, but a mile high. Now you only need to worry about positioning the cursor horizontally, not vertically, so the task of clicking on a menu item is that much easier."[14]

JavaFIBS is a well written program. Some, but not all, of its flaws lie inherently with Java. JavaFIBS ability to let users swap custom boards, its ability to mark friends, stability, and its robust preferences make it s very fine client. However, compared to

CocoaFIBS it just does not hold up as a visually pleasing and intuitive application for Mac OS x.



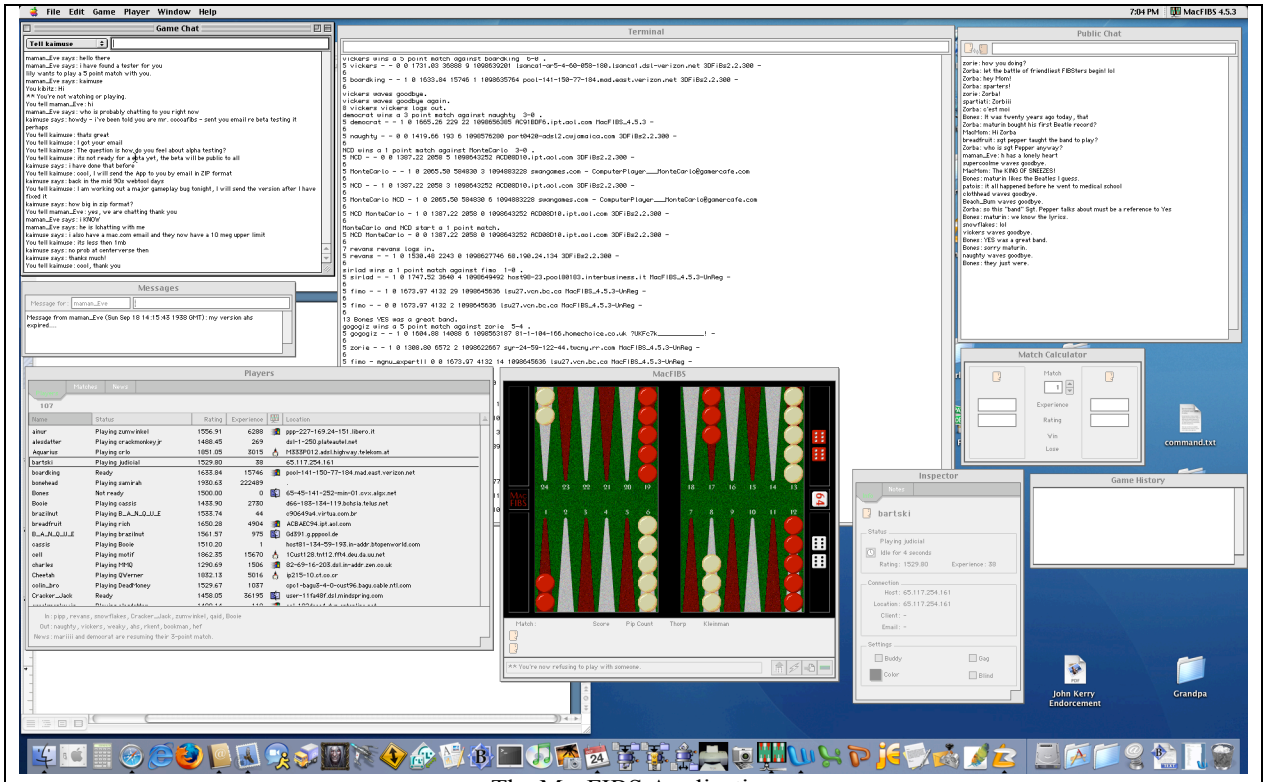
The JavaFIBS Application

4.2 MacFIBS

MacFIBS is the original Macintosh FIBS client. Unfortunately, this program only runs on Mac OS X. Apple's Mac OS 9, which like Windows, contained years of what professor Joseph Bergin likes to call "code smell". "It is just a bit of code that doesn't seem right to you, not because it is necessarily a bug, but because you know you can do better if you think a bit harder. Code with consequences. A more extreme definition would be a pet "accident" on the living room rug. You can still live with it, but life would be much better if you get rid of it." Code smell is the result of years of older code from previous versions being re-used. Since Apple re-wrote the Macintosh Operating System from the ground up, all older applications would not run natively. To accommodate users in the limbo stage of switching, Apple developed "Classic" an emulator similar to Microsoft's Virtual PC that emulates Mac OS 9 inside of Mac OS X. The drawbacks of software emulation are well known. Emulated software will always run slower and with less compatibility in general with code written natively.

MacFIBS' other major drawback besides running in emulation is the amount of windows that it uses. CocoaFIBS combines the game window, private chat, public chat, game chat, console window, and player list into a single window. MacFIBS uses five different windows to accomplish the display of the same information. This is not entirely due to poor design. A huge paradigm shift took place in the Mac OS development world during the transition from OS 9 to OS X. Better and more elegant window management became a core tenet of the Aqua User Interface. Sheets, drawers, tabs and streamlined

windows all contributed to this revolution. With the exception of these two drawbacks MacFIBS is an extremely robust and well written client.



The MacFIBS Application

5. Conclusion

5.1 Release Plans

CocoaFIBS is currently in Alpha Release status. A select group of real FIBS users have been helping me to trouble shoot, debug, and recommend changes for the application. Feedback and enthusiasm have been very high. “Keep up the great work... it's a service to all mankind” (Jason Hall, jason@jasonjhall.com). “Oh, please, oh please, can I help test Cocoa. I confess I am not computer saavy, but I'm determined to avoid java at all costs. I adore MacFibs and am excited about a [Mac OS X] alternative. Can I, can I, please?” (feklhr@earthlink.net). I will keep CocoaFIBS in alpha release for another month. This process will enable me to continue to get feedback and suggestions as well as debugging.

5.2 Future Work

After my thesis is complete, I plan to continue development and support of this project. I will set up a web page, conduct public beta testing and plan to have a public release by mid 2005.

6. References

1. Backgammon Book by Oswald Jacoby
2. http://www.wordiq.com/definition/Backgammon Computer Backgammon
3. http://www.fibs.com/guide.html#history
4. http://www.dekorte.com/Objective-C/
5. http://developer.apple.com/cocoa/
6. http://www.bignerdranch.com/
7. http://macedition.com/feat/feat_bnr_20020424.php
8. http://www.fibs.com/fibs_interface.html
9. IBID
10. http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/
11. http://www.joelonsoftware.com/uibook/fog0000000249.html
12. http://developer.apple.com/documentation/Cocoa/Conceptual/Drawers/index.html#//apple_ref/doc/uid/10000001i
13. http://www.joelonsoftware.com/uibook/chapters/fog0000000063.html
14. http://www.joelonsoftware.com/uibook/chapters/fog0000000063.html