Multi-Player Cribbage

by Travis Reese

Introduction

I first came to Northern Michigan University in fall of 2008. I wasn't sure what I wanted to study, but it was suggested to me to become a computer programmer. My father has been a programmer for nearly 40 years, and taught me a lot about computers as I was growing up. From an early age I learned about hardware, and then scripting languages like MS-DOS shell scripting; and by high school I was building my own computers and writing simple programs. So it seemed like an easy choice to seek a degree in Computer Science. Over 7 of the last 11 years I have taken classes that have brought me here to my Senior Capstone Project.

For this project I wanted to do something involving networking since that's always been an area of computing I've liked best. I also wanted to include some form of research in my project since I enjoy learning new things. This brought me to Java networking since Java is likely the language I will use in industry; and while I have learned networking in many languages throughout my academic career I never spent much time on Java. I chose to build a game application because it was a straight forward way to incorporate desktop applications with client-server networking.

Project Overview

This application is Java based Cribbage game. It uses JavaFX as a client-side Graphical User Interface; and Java networking as it's communication backbone. On the server I used a inline network server and game host, with separate objects for cards and players. For network communication I created objects that allowed me to send what ever information I needed. While this project was not entirely object-oriented, I used many principles behind the object design schema in its creation. While none of the data structures or algorithms were overly complex, due to the nature of the project I did spend a fair amount of time in the design stage to ensure that.

Program Features

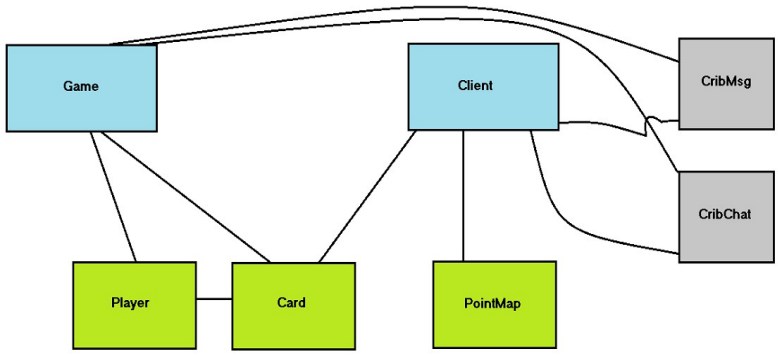
Some interesting feature of my project include:

- Dual server design
- Multi-Threading
- Multi-Window JavaFX Client
- Custom Network Objects
- In-Game Chat

I was able to design my server in such away that a single program not only runs the game, but is host to two separate network servers. The first server is for all game communication and the second is for the in-game chat feature. The in-game chat was something I wanted to include, but hadn't yet thought about how to implement when working on the game networking. While creating a network model to merge in and attempting to resolve other network related issues I had an interesting revelation. I was able to implement the chat function with minimal effort due to the nature of my network design. In that design I

was able to multi-thread both my client and servers. While designing my communication scheme I decided it was best to create a few objects that allowed me to send game data, messages, and even the card objects.

In terms of how my code has been organized and structured, I took a approach somewhere between object design and functional. I have 7 object classes:



I think this structure worked well for it's purpose, but if I could start over I would have separated the Game class into 3; one for the game and 2 separate network servers for the game and chat functions. And I may have also changed the client and game to be more object oriented.

Difficulties

While working on the research portion of my project I discovered that I had poorly gauged the comparative difficulty of JavaFX and Java Networking suite. While I knew nothing about JavaFX finding resources to explain it was rather easy. JavaFX is also very well documented, and as I read through that documentation I discovered many similarities to Java Applets which I learned as a freshman. On the other hand the documentation for Java Networking was less helpful, and I scoured many sites looking for information on setting up a

network in java and found very few comprehensive sources. JavaFX is also easy to find video tutorials on while Java Networking has almost no usable video tutorial sources. Luckily after reading enough of the Oracle documentation a little help, as well as some trial and error I was able to piece it all together.

When working on the networking I ran into some trouble relating to how to send data. In the end I used the ObjectStreams rather than the DataStreams because it gave a simpler interface and allowed more information to be sent. Once I had gotten to that point I had further issues with stream initialization lock. It turns out that if you try opening the input streams on both ends it will block and wait indefinitely without ever setting them up. And furthermore the data appeared to never actually send once written to the stream. The solution turned out to be starting the outputStreams first, and then using a flush() after each message send to ensure the data was written and not just buffered.

Once I had the networks working and communicating, and learned something by researching an error. In order for objects to be sent with any accuracy, they must be serialized as a data integrity check. That mean serializing all the objects I had to send.

I ran into an issue related to networking when working on my client. I learned that you cannot block the GUI thread while waiting on network traffic. The solution to this ended up being multi-threading. This not only solved the problem with the locked GUI thread but provided an easy method for implementing a chat feature. By reusing existing code, and extending the new multi-threading feature I was able to create a second inline network server to handle chat functions by spawning a new thread for each client.

I also discovered a minor error relating to pictures where relative pathing didn't work and the file load completed without error and without loading the images. Unfortunately the only solution I was able to find was to use absolute pathing.

A Look Back

Looking back over the project I would say that the most difficult part was keeping my project to a time table. Finding hiccups in certain areas and having some things work easier than expected only to have issues down the line, meant that my initial time estimates were hard to stick to. Although the hardest technical issue was figuring out the networking. I didn't anticipate having to create a side project and model the network in order to be able to set it up within my project.

Overall I would say my project was similar to what I expected as far as difficulty, perhaps a little harder. Fortunately most of my issues along the line were small issues cropping up that took more time to solve, and not systemic issues that ruined the project.

If I were to change my approach to this project having to do it over again, I would certainly have used a more test driven approach. Since my biggest issues were time issues, test driven approach may have reduced those setbacks. I also would have utilized the multi threading more, as well as pushed for a more Object Oriented design than I did. Last, I would have considered a different card game like Euchre with more straight forward rules. That also would not have had a board, and allowed me to do more card effects. I would have also liked to do more on the research aspect.

Conclusion

In conclusion I think my project went fairly well in the end. I had to put in some hard work, and deal with a lot of time delays, but in the end I feel it was a worthwhile experience.

As of completing my project I have done the following:

- Research
  - Java Networking (5)
  - JavaFX GUI (10)
- Server Application (25)
- Playable Game (10)
- Client Application (20)
- JavaFX GUI (10)
- In-Game Chat (15)

This brings my total points (per my proposal) to 95 out of the possible 135.

I would also like to thank the professors of the Math and Computer Science department of Northern Michigan University for their dedication, and enthusiasm in their pursuit of education.