ONLINE CARD GAME

Two player online German Whist

Arunjit Singh

Summer 2009

ONLINE CARD GAME

Two player online German Whist

Arunjit Singh
GGSIP University, Delhi
Summer 2009

DECLARATION

I hereby declare that all the work presented in this document titled "ONLINE CARD GAME" is original and is an authentic record of my own work done during the period of July 1, 2009 and July 20, 2009.

Arunjit Singh

TABLE OF CONTENTS

INTRODUCTION

Online Two Player Card Game [German Whist] is an implementation of a card game played between two players, and in this case, over a network connection.

The objective of this project is to create a useable implementation of a slightly unknown, albeit fun card game, German Whist. The game is played between two players using a single card deck.

The two main parts of the project are the server and the client.

The stand-alone server opens ports and accepts connections to a socket. It includes a `ClientHandler` class to manage message flow from a `Client`. The server also includes a `GameHost class`, which is the virtual 'table' on which a game is played. The server may use any number of tables that can be supported.

The server also includes an implementation for the cards and card decks. The card deck for every game is managed by the `GameHost` on the server. This prevents users on the client applications from having any influence on the card decks, and hence the game. The rule set for the game is implemented on both the client and the server. The scoring is managed by the `GameHost`.

The client functions as an interface to the game. It manages user log in to the server, an output console and a Canvas on which to play the game. That is, it is the graphical implementation of the `GameHost` (`CardTable`) and `Card` on the server, and introduces client-only requirements of `Hand`.

About the game

This game is a type of whist for two players. The player with the most number of tricks at the end of a game wins.

Whist is a classic trick-taking card game. A trick is a single play by each player, and is won by the player having the highest card.

In German Whist, the game is played in two halves. In the first half, the players try to score a better hand for the second half by playing for the face-up card. In the second half, the players play to win more tricks.

About the platform

Java refers to a number of computer software products and specifications from Sun Microsystems that together provide a system for developing application software and deploying it in a cross-platform environment.

Java compilers produce byte code from the source and an interpreter, the Java Virtual Machine (JVM), interprets this to an executable program. More information about Java is available on Sun's website.
(http://java.sun.com).

Java was chosen for this project primarily because of cross-platform availability. This software was developed on a Macintosh (OS 10.5) and tested on Windows (XP), without having to vary the code (more on testing and hardware specifications of test machines later).

System Specifications

Minimum

- Any computer running JRE 1.5 or better
- 1.8 GHz processor
- 512 MB + RAM
- At least 10 MB Disk space
- Display resolution of 1024x768 or greater
- An Internet or LAN connection required

Recommended

- A computer running JRE 1.5 or better
- 2 GHz + Core 2 Duo processor or better
- 1 GB + RAM
- At least 10 MB Disk space
- Display resolution greater than 1280x800
- A broadband internet or LAN connection required

THE PROJECT

Project Aim

A user friendly application for playing a card game called German Whist. The game includes ability to connect to a server, log in/out and play against a player over a network.

Conceptualization

The idea of the application is simple. A user must be able to send and receive information from a server. The server starts up a game for the user and handles messages and plays.

Connectivity

The users connect to the server using sockets. This allows for full control over the connectivity between a user and a server.

The server accepts all connections on a specified port and sends the client socket to a handler for the client (`ClientHandler`). The client handler manages the input from a client and sends appropriate method calls. Once a user has successfully connected and request a table, an object of the `GameHost` class adds the user if it has any spots available. Adding to a table is done on first-come first-serve basis. Once both users are connected, a game starts. The game is turn based, and the first player to join at a table plays first.

Game-play

The basic game-play of any card game is the playing of cards. Whist requires only a single card played per person, per turn. Since the server manages the decks and the game-play decisions (as the game table), it essentially acts as the dealer and the controller. Therefore, it only needs to know the card played by a player. So, the only messages required to be passed are the cards.

A card has two basic components: a rank, and a suit. Playing cards are divided into 4 suits of 13 ranks each. The rank corresponds to a face value for a card (which may differ based on the game played). The suit and the rank can be combined to form a single value for each card, which can be passed over the network between a client and the server.

Example:
If “spades” has a value of 400, and an “Ace (high)” a value of 14, the value of the card becomes 414. This value can be deconstructed to create the original card:

[Code:]

```
(int)(414/100)*100    =    400        “spades”
(int)(414%100)        =    14         “Ace (high)”
```

The values of suit and rank can be used to display the card for the user. This approach removes the requirement of serialization. Also, this method is simple enough to be reused.

User’s Interface

The user’s interface comprises of a window on which the game is played, a log in/out screen and a console.

The user interacts with the cards in ‘hand’: a ‘hand’ is a list of cards the user has. For this game, the user can directly interact only with these cards. The `Hand` is displayed across the bottom of the user’s window, with the discarded cards on the left and the stock pile on the right. When it’s the user’s turn, he may click on the card he wished to play. The interaction with the table is disabled when it is not the user’s turn to play.

Programming Concepts

GUI

The Graphical User Interface is implemented using `javax.swing.JFrame`. All other components are added to this.

Connections

The connections are created using two classes from the `java.net` package: `Socket` and `ServerSocket` (for the server). These run within a `Thread`, enabling multi-threading features which greatly improves performance. Multi-threading allows many tasks to run simultaneously as individual threads.

Client-Server IO

The message passing between the client and the server is done through `BufferedReader` and `DataOutputStream` (`java.io`).

Game UI

On the client, the display of the cards is on a `Canvas` component. Each `Card` object stores its own image as an `ImageIcon`, which can be loaded before drawing.

CLASS DESCRIPTIONS

Server

ajs.gws.server.GermanWhistServer

The main server. This class opens ports and accepts connections in a thread. It is also used for sending messages to players.

ajs.gws.client.Client

This class is used to manage client connections on the server. It stores each client's name, socket port and other relevant information. It also includes a reference to the `GameHost` class when a user joins it.

ajs.gws.client.ClientHandler

The handler for the client messages. It runs as a separate thread to receive client messages.

ajs.gws.GameHost

The host for each game. This is where the game is actually played. It includes a rule set for the game, starts and stops the game and manages game-play.

Client

ajs.gwc.GermanWhistClient

The main client application. It includes all the components that enable the user to log-in and play the game.

ajs.gwc.CardTable

This models a card table. The cards appear in this component and the user interacts through this. Includes basic play checking.

`ajs.gwc.LoginWindow`

Allows users to connect to a server.

`ajs.gwc.Console`

An output console for the application.

Common Classes

`ajs.cards.Card`

Models a card. Includes methods to allow it to draw itself only when required.

`ajs.cards.CardConstants`

Ⓘ An interface that includes constants used to model cards.

`ajs.cards.CardDeck`

A model of the card deck. Instantiates the required number of cards for every game. Available only on the server.

`ajs.cards.CardStack`

A utility that models a stack of cards. It is used where-ever there is a requirement of a set of cards where only the top card may be used. This is used for decks and discard piles.

FUNCTIONALITY

Flow

I. The server is started on a specific port (default: 5335).
II. The client connects to this server using its LAN or internet address and the same port number. The user enters this data along with a user-name in the log in window.
III. If no-one else by that user-name is connected, the server accepts the connection and adds the new user. Otherwise, the connection is rejected and the user must enter a different name.
IV. When a connected user requests a table, the table's availability is checked and the user added if a place is available.
V. When two users are added to a table, the game starts and the cards are dealt. That is, the `GameHost` sends the clients messages pertaining to what card was drawn and dealt.
VI. The players play the game by clicking on the cards that appear on their screen.
VII. When the game ends, the player with most points wins.

A graphical representation of this flow is given in the next section.

Flow Diagram

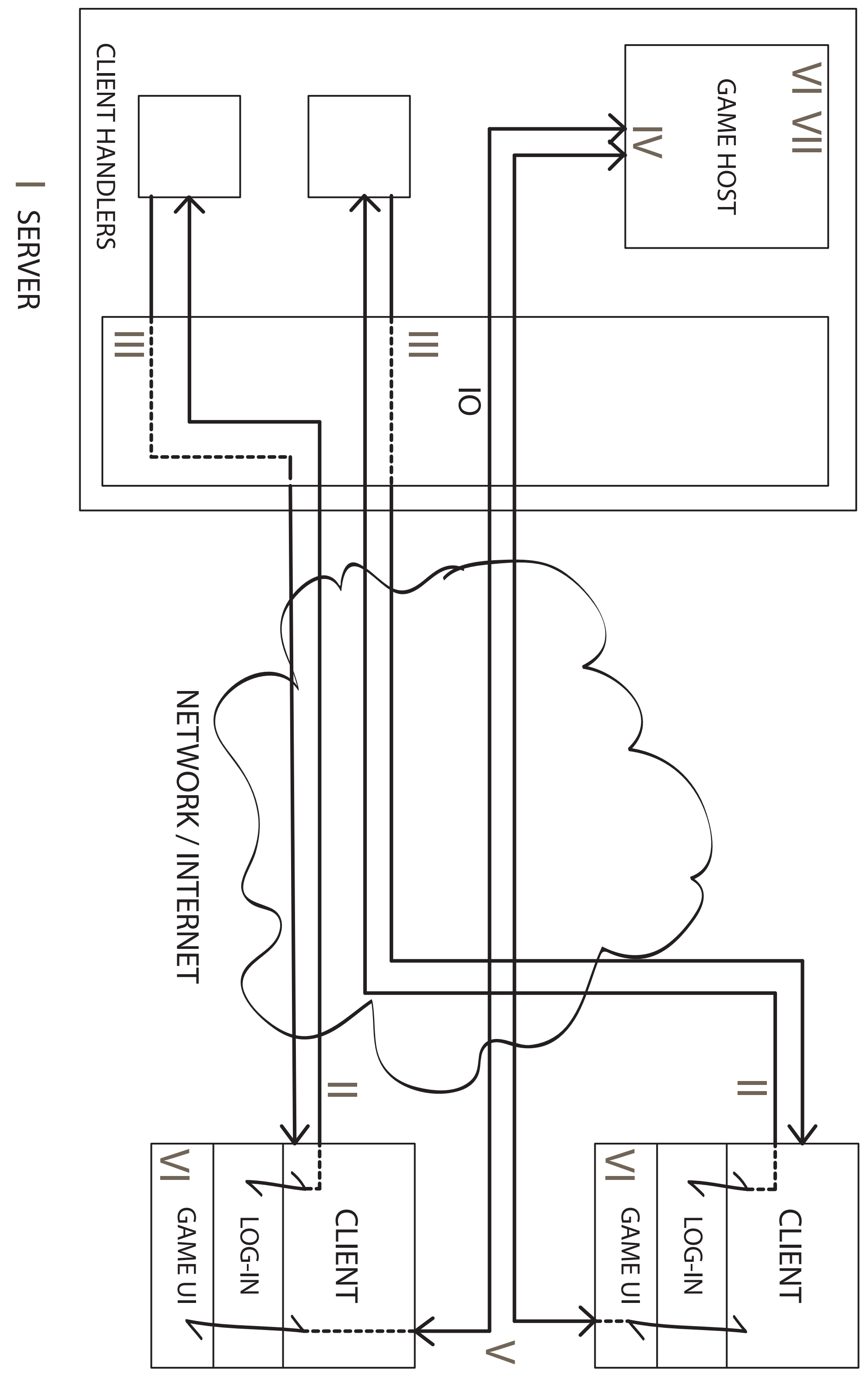

Diagram showing the flow of the game from logging in, to game play and finish.

TESTING

Test Machines

The primary development of the software was on the Macintosh platform under Mac OS 10.5. The applications were thoroughly tested on other machines with the following specifications:

Mac OS X

- 2.2 GHz Intel Core 2 Duo processor
- 2 GB RAM
- JRE 1.5 and 1.6
- 1440 x 900 px display resolution

Windows XP

- 1.8 GHz Intel Centrino processor
- 512 MB RAM
- JRE 1.5 and 1.6
- 1280 x 800 px display resolution

And

- 2 GHz Intel Pentium D processor
- 384 MB RAM
- JRE 1.5 and 1.6
- 1024 x 768 px display resolution

Tests

The software was tested for the following:

- Cross-platform execution
- Stability
- Client-Server IO
- Game-play and UI

Cross-platform Execution

Expected Results	:	No errors
Actual Results	:	No errors

Remarks:

The applications work across platforms. For gaming, this is a huge bonus.

Stability

Expected Results	:	No errors
Actual Results	:	No errors

Remarks:

The applications work without crashing. They are stable.

Client-Server IO

Expected Results	:	No errors
Actual Results	:	Error — Random order of messages

Remarks:

Due to the multi-threaded nature of the server application, the messages may be actually sent not in the order that they were programmed. This is machine and OS dependant, and was a concern.

The error was rectified by introducing a delay in the thread when a message was sent to allow previous messages to be delivered.

Game-play and UI

Expected Results	:	No errors
Actual Results	:	Error — Flickering of the graphics on lower-end machines.

Remarks:

Due to lesser memory available, the lower-end machines had a noticeable delay in updating and repainting the display. This lead to flickering.

The AWT runs its displaying on a thread which repeatedly updates and repaints the display. This is memory intensive and was required to be addressed.

The error was rectified by overriding the update method and calling repaint only when the user played (clicked on) a valid card, or when the program required to add a new element on the screen.

Error rectification

The errors identified in the previous section were rectified as stated and the application was re-tested for these problems:

<u>Client-Server IO</u>

Expected Results	:	No errors
Actual Results	:	No errors

Remarks:

All messages between the client and the server were passed as expected. There were no more errors in these.

<u>Game-play and UI</u>

Expected Results	:	No errors
Actual Results	:	No errors

Remarks:

The flickering of the display was removed and the game could now be played smoothly.

CONCLUSION

Current Versions

The current version of the project includes an easy interface for the user to play a single game against a networked opponent. The server provides basic functioning for enabling online play. It's object-oriented design would help further improvements and has scope for upscaling. The client program is limited to one game per run.

Future Versions

Future versions of the project have some good groundwork to work from. On the client side, fully functioning multiplayer gaming can be enabled.

What's Expected

Server:

- Stored user preferences. Allow user to override defaults.
- Multiple games' support. Run various other card games on the same server.
- Better messaging system to enable easier development of other card games.

Client:

- Stored user preferences.
- Auto join available server.
- Off-line gaming against a computer opponent.
- Challenge other connected users.
- Scalable graphics for higher resolution displays.

REFERENCES

- Extensive use of the Internet (Google, Wikipedia).
- Sun Microsystems' Java documentation pages (http://java.sun.com).
- GNU (http://www.gnu.org).

APPENDIX

Server-Client Messages

In an effort to maintain uniformity and a generic form for future development, all messages passed between the server and the client have six characters, followed by a colon.

Server to client

Message	**Meaning**	**Example**
ACCEPT:	User was accepted	ACCEPT:Welcome!
REJECT:	User was rejected	REJECT:User exists!
ADDUSR:	Add a user	ADDUSR:username1
START1:	Game starts	START1:Game started
INSERT:	Insert a card to the player's hand	INSERT:414
OPENED:	A card was opened	OPENED:312
TURNS1:	This player's turn	TURNS1:Your turn
TURNS0:	Not this player's turn	TURNS0:Not your turn
PLAYVS:	Opponent's card	PLAYVS:102
WHIST1:	Whist started	WHIST1:Whist started
LOGOUT:	A user logged out	LOGOUT:username4
JOINED:	A user joined a table	JOINED:username2:1
ERRORS:	An error occurred	ERRORS:No Connection
NEXTPL:	Next round is about to start	NEXTPL:Next play
WINNER:	This player won	WINNER:8-5
LOSER!:	This player lost	LOSER!:5-8

Client to server

Message	**Meaning**	**Example**
NEWUSR:	A new user request	NEWUSR:username1
LOGOUT:	User is logging out	LOGOUT:username2
JOINAT:	Request to join a table	JOINAT:1
PLAYED:	A card was played	PLAYED:211

Given Card Values

Suits

Spades = 400
Hearts = 300
Diamonds = 200
Clubs = 100

Ranks

Ace = 14
King = 13
Queen = 12
Jack = 11
All other cards have the same value as their face (like Two = 2, Three = 3, etc.).