Husky Holdem Statistics Generation Detailed Design Specification Extract

Version 2.0, 12/1/2008

Aaron Cook, Chris Revel, David Chanko, Dan McNamara

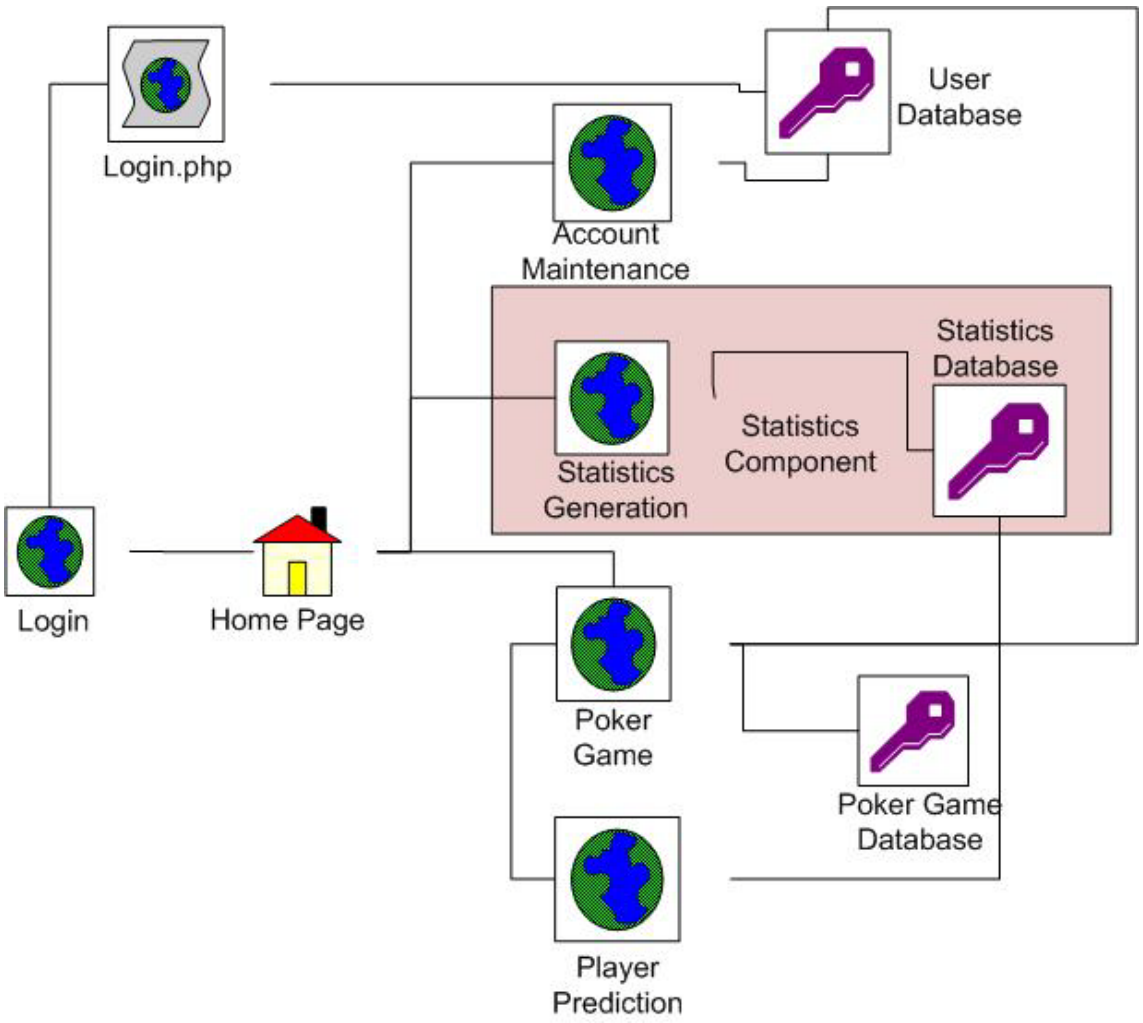
0.0 Purpose

The purpose of this document is to describe in detail the Statistics Generation Component of Husky Hold'em. The Statistics Component interfaces with several components - an interface to upload and download hand histories, a statistics viewing area where users can examine different measures of their past performance, and a graph generation interface to provide the users a visual representation of the data collected.

1.0 Component Definition

The overall purpose of the Statistics Component, highlighted in red below, is to provide the user with statistics from either uploaded hand histories or from games played on the Husky Hold'em website. The statistics are accessed via the home page of the website.

Whether these statistics are coming from uploaded hand histories or from locally played games, the statistic data is retrieved from the Statistics Database. Once these are retrieved, they are then converted into various statistics and graphs.



2.0 Component Interface Specification

2.1 Interface Purpose and General Description

There are two primary interfaces for the Statistics Generating component. The first is the in-game user interface. Its purpose is to provide relevant information to the player during a hand. It will use Javascript, AJAX, and PHP to connect to the game database, pull preselected game and player statistics from that database, and display them on the game screen, using AJAX to sidestep the need to constantly refresh the active web page.

The second interface is the detailed hand history. Players will be able to search through a history of hands played, based on several criteria, to determine their past performance and the likelihood of a favorable outcome given a player's current hole cards. Additionally, players will be able to determine general behaviors of opponents, based on previous hands that they have

faced each other in. This interface is written as a web portal, using Javascript, AJAX, and PHP to connect to a MySQL database. The database is currently running on the same server as the web server, however, it can be easily offloaded to a dedicated database server.

2.2 Communications Method and Interface Definition

2.2.1 Overview

Communication between the different subcomponents in the Statistics Generation component is accomplished chiefly though parameter passing via HTML's POST messages. Due to the browser based nature of our system, to pass messages containing user requests to the server it is necessary to POST the messages to the server when the call to a specific HTML file is needed. On the whole, there are six different interfaces over which these POST messages are passed, one for each of the principle PHP Logic files discussed below. These PHP files use basic logic to interpret these messages, or the lack of these messages, and then execute different procedures based on the results.

2.2.2 Communication Method

The general structure to formulate the communications is as follows. The first part of the message consists of an opt or option code that indicates to the PHP file what sub-function needs to be executed. On the receiving side, the PHP file checks the POST array provided and examines the value there. Depending on the value a branch associated with a particular function is taken. After the option code comes the parameter codes. These are indicated by a two character identifier code followed by an index to denote the order of the parameter. The PHP file checks the POST array for these parameters and either stores them in a list to be passed to the function later, or, if the function takes only one of the parameters at a time, calls the function once for each of the parameters passed. It is not necessary to specify the number of parameters passed since the PHP function will iterate through the parameters until it finds a parameter that is null or undefined at which point it will stop.

The protocol used to return data to the browser is automatically handled by the HTML protocol. All the PHP function needs to do is output the data using the printf() call, or the echo statement and this information will be sent to the user's display.

2.2.3 Interface Definition

Overall, the communications method is very simple and follows the basic structure of a PHP file, an option code, followed by a list of parameters, which may be empty or have one or more values which the PHP file interprets and uses to choose a function call to execute. The data is returned automatically using the HTML responses generated by the text output by the PHP file. A table is presented below to detail the generatestats.php file's message interface for two of the statistics offered. The third entry in the table demonstrates how to request multiple statistics at once. All other PHP file interfaces can be specified in a similar manner.

PHP File	Options	Function	Parameters	Description
generatestats.php	stat1=<stat code (i.e. mot)>	MoneyOverTime()	hh1 ... hhn=<List Hand Numbers>	Requests one statistic from database, generated from data specified in the hand history numbers list. If list is empty, generate over entire database.
generatestats.php	stat1=<stat code (i.e. sdnp)>	ShowdownPercent()	hh1 ... hhn=<List Hand Numbers>	Same as above.
generatestats.php	stat1=sdnp&stat2=mot	ShowdownPercent(), MoneyOverTime()	hh1 ... hhn=<List Hand Numbers>	Same as above except results for two statistics are generated over the same set of hands.

While the above deals mainly with the communication between the user interface and the back end logic, the second option for communication involves communication between the different PHP Logic Layer functions. To allow the functions to communicate with each other over long periods of time, SQL is used to preserve the information in a database. For example, SQL is used by import.php to store data in the statistics database so that later gethistory.php and getstatistics.php can access the data and display the information to the user.

2.3 Specific Interface Design

As discussed above message passing over HTML POST is the primary method used to pass data between sub components of the Statistics Generation component. This section goes through each of the six PHP Logic files and describes in detail the different message options, how they are interpreted, and finally how they are translated into function calls. We will not go into detail regarding how the data is returned to the user, as this is handled automatically by the HTTP protocol, as discussed above. The details of the interface are similar for each of these files and as a result only generatestats.php is defined precisely. All other PHP files follow the same pattern.

search.php

Search.php is fairly simple, as there are only three basic search criteria areas that need to be displayed to the screen. The message passed to this component is of the form "opt=x" where x is an integer from 1 to 3 denoting what functionality needs to be printed to the screen. The corresponding function calls associated with the opt number are as follows:

- 1: printOpponentSearch(): This outputs HTML which allows the user to specify parameters about who their opponents were in a particular hand.
- 2: printTableSearch(): This outputs HTML which allows the user to specify parameters about the layout of the table in a particular hand.
- 3: printHoldSearch(): This outputs HTML which allows a user to specify parameters about the hold cards they were dealt in a particular hand.

An example message passed to search.php would be "opt=1" indicating that the HTML to set parameters for an opponent search should be printed to the screen.

statistics.php

This sub-component has some of the more complicated messages passed to it. There is one parameter for each of the three search option categories. These parameters are checked when they're passed and a SQL statement is created based on what parameters are have values for them and which parameters are null. To help ease the burden on the server, JavaScript is used client side to examine the user's selections and create parts of the query that will eventually be used to get the search results. These sub-statements are then passed to the server and assembled into the main SQL statement. There is only one function which needs to be called after the SQL statement has been created, and this accepts only one parameter, which is the SQL statement. Because of this, once the SQL statement is completed, it simply needs to be passed to the function and executed to find the appropriate results.

import.php

This is perhaps the simplest of the six in terms of message complexity. When this function is called, the file selected by the user is uploaded to the server, where it can be found in the PHP $_FILE array. The only data that needs to be passed in the message is the name of the file. Since there is only one parameter, there is no need to decide what function to call, the file is simply opened, the parsing is executed, and the results are stored in the database. An example message passed to this file would be "file=uploaded_file".

export.php

This file takes in a list of hand histories to export to a file. This function receives a list of one or more hand history IDs in the form of a list such as the following: "hh1=123456&hh2=678909&...&hhn=2584845". Unlike most of the other functions, it is not necessary to first check this information before using it, the function simply iterates through the list of values and calls gethistory.php for each hand history ID.

gethistory.php

This function is not actually communicated with through HTML POST like the other methods, but is instead called directly from export.php and statistics.php. The function call is quite simple and takes two arguments, one denoting a file to write to, or a null value if the results

are to be written to the screen, and a second value to identify the ID of the hand history to retrieve.

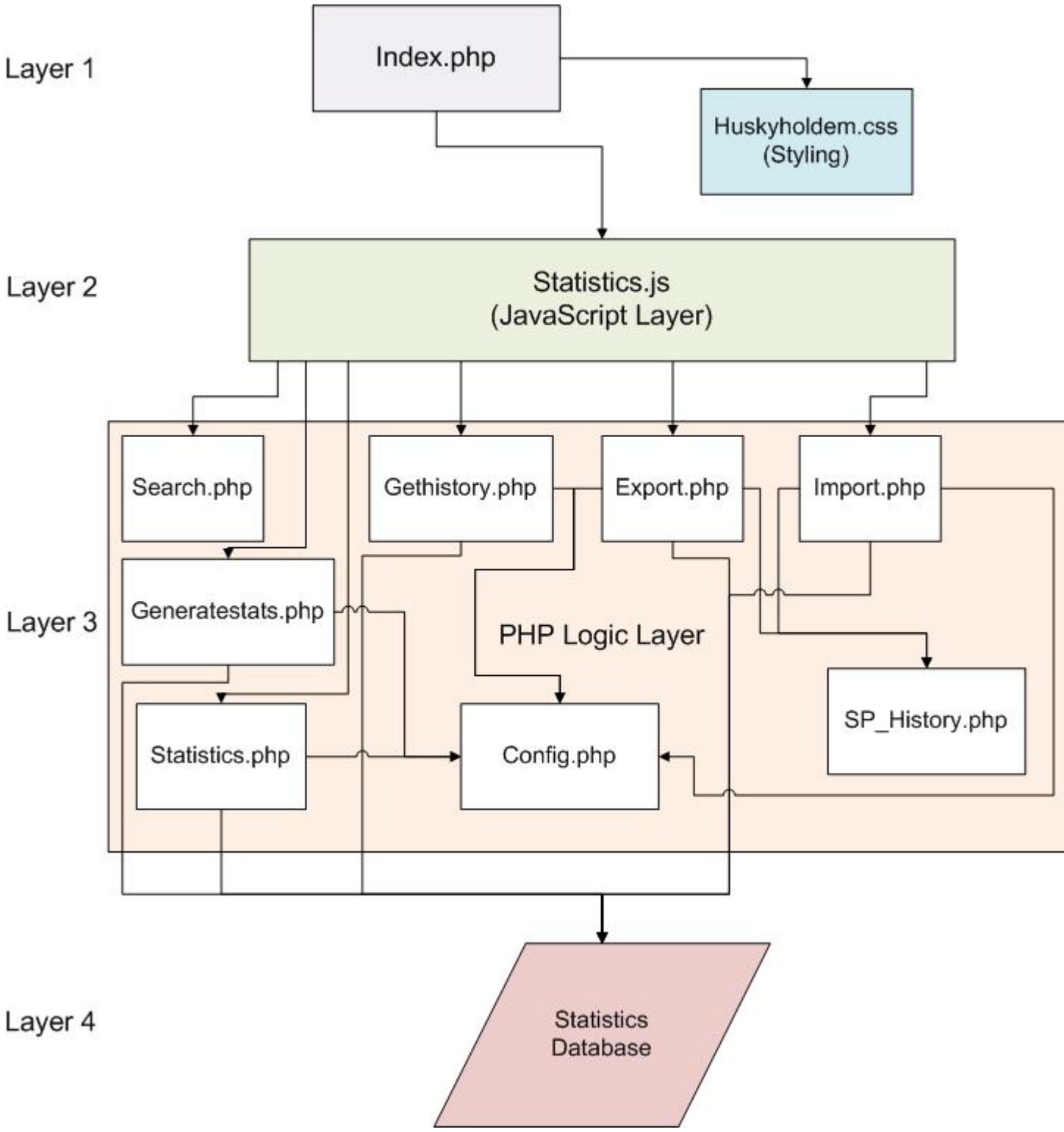
generatestats.php, etc.

This function requires the most detailed message passing due to the complexity of passing multiple statistics to be generated along with their parameters. Nevertheless, the same basic pattern can be followed and the results is a list of the form "stat1=mot&stat2=shdwn&hh1=137048&hh2=7492565&...&hhn=54237542" where stat1 and stat2 represent different codes of statistics to generate, and hh1, hh2, ... , hhn represent the IDs of hand histories to generate the statistics on. Generatestats.php interprets these messages by looping through the hhi list and recording the games that need to be considered in the statistics. The stats values are then looped through and the game list is passed to the statistic function associated with each stat code. This function returns an HTML string which is then added to the overall HTML response that is eventually passed back to the user. The user can request a textual, tabular, or graphical form for the results. Textual form is just a list of the results, tabular form presents the data in a table where appropriate, and graphical generates graphs for appropriate statistics. Graphical is selected by default. Details of the arguments, parameters, and data types for this interface are provided below. The message parts for all the other PHP files can be specified in a similar manner.

Parameter	Data Type	Sample Values	Description
stat1 ... statn list	String code	mot (Money Over Time), sdwn (Percent of hands that go to a showdown)	String code denoting a list of statistics to generate.
hh1 ... hhn	Integer	12345678	List of integers denoting database ID of a hand history to generate statistics from.
display	Integer	0 (Textual), 1 (Tabular), 2 (Graphical)	Determines how the results should be displayed to the user.

Aside from the interfaces discussed above, we have our data model to keep track of all this information. The rest of this component interacts with this database though SQL statements which are used to import data into the database, export data from the database to a text file, and also to generate statistics based on the data for presentation to the user.

3.0 Component Design


Layer 1
Index.php
Huskyholdem.css
(Styling)
Layer 2
Statistics.js
(JavaScript Layer)
Search.php
Gethistory.php
Export.php
Import.php
Layer 3
Generatestats.php
PHP Logic Layer
SP_History.php
Statistics.php
Config.php
Layer 4
Statistics
Database

3.1 Overview

The general design of the Statistics Generation component is as follows. The system is divided into four layers. The first layer consists of the user interface where all the data presentation and user interaction occurs. This layer consists of the main php file, the user interface, the images used in the site design, and the style sheets used to form the site's layout.

The next layer acts like a bridge between the first and third layers by recording or returning data based on users' interactions with the presentation layer. This layer is necessary to allow the site to take advantage of AJAX technology to update the user interface. AJAX allows the browser window to be updated without the page needing to be refreshed. This is accomplished by using javascript to call php functions which return dynamically created HTML text, which is in turn inserted into the presentation layer. The javascript described above makes up the second layer, and the php files it calls comprise the third layer.

The third layer consists of all the code and logic to handle the functionality the second layer provides the presentation level. These PHP files are not accessed directly by the user, but are called as necessary from the second layer javascript.

The last layer is the data layer, consisting of our statistics database, php functions which contain sql routines to insert or view table data, and configuration files which hold parameters for connecting to the database and information about the web server.

To show how these layers interact, we describe the series of actions which respond to the user requesting a set of hand history files based on a set of search parameters. First, the button triggering the search in the presentation layer interprets the action and calls the appropriate function in the JavaScript layer. The JavaScript function interprets the parameters passed by the HTML page and determines that statistics.php needs to be called and proceeds to record all the search parameters entered by the user which are then encapsulated in a parameter list and sent to the statistics.php file on the server. Here the parameter list is organized into a SQL query that is passed to the statistics database where it is executed and returned to statistics.php. The PHP file takes this list and creates HTML text to display the information to the user and allow the user to view the hand history files returned in greater detail. This HTML response is then passed back up to the JavaScript layer where it is inserted into index.php for the user to view.

In the following sections, we discuss in detail the different sub-components that make up each of these four layers.

3.2 Presentation Layer

index.php

This is the first file the user encounters when entering the website and contains a general interface from which the user accesses the functionality of the component. The code for this section is quite simple and only needs to provide the user with the option of searching for hand history files based on three general criteria: opponent information, table layout parameters, and hand data. This is accomplished by providing links for the user to select. When clicked, the action is interpreted by the JavaScript layer and the appropriate html is fetched from

search.php and is displayed on the screen where the link was displayed. Once the user is done filling in the criteria, there is a submit button which takes all of the parameters and passes them to the statistics.php file which formulates a SQL query, executes it, and prints the results to the screen. Aside from this data is another link that allows the user to hide the search options. Again, this portion is very simple and contains just enough information to allow the user to begin the searching process.

huskyholdem.css

This file contains the styling information to display all of the html elements associated with this component. This includes the tables to display the data, styling for links, and helps the statistics component blend in with the rest of the system.

3.3 JavaScript Layer

statistics.js

This is an extremely important component, allowing the user to view all of the dynamically generated HTML files from the PHP Logic layer. With AJAX, rather than reloading the page for every user action, only certain areas of the web page need to be updated at once. This requires some overhead to take in user's actions, interpret them, request information from the PHP layer, and finally output the resulting HTML text to the screen. This JavaScript represents this overhead, which is implemented using the following functions.

AJAX Functionality:

This section contains the general functions that are needed to execute take advantage of AJAX.

void getResults(integer opt, integer list);

This function is called when ever a user clicks on any link or button on the site. The user's click calls this function and passes an option, *opt,* and an optional parameter *list* which contains other parameter information if necessary. The function executes the following general procedure with very little variation:

- Create an HTTP object, which is used to connect to the server and call the necessary PHP file.
- Then the option parameter is interpreted, which includes the following actions:
    - Set the URL for the necessary PHP file.
    - Compute any parameters, if necessary.
    - Set the function which is executed when the XML response text is returned.
    - Send the Request.

There are six different versions of this general procedure contained in this funciton, one for each of the key PHP files discussed below: statistics.php, export.php, search.php, gethistory.php, import.php, and generatestats.php.

void stateChanged()

This function is called whenever the browser receives anything after making a request the server in the *getResults()* function above. This function simply takes the response text, finds the appropriate container on the web page, and sets that container's text to the HTML text received. This function represents the second half of the AJAX data retrieval process.

XmlHttpObject GetXmlHttpObject()

This function simply creates the XmlHttp connection object used by *getResults()* to send the request to the server. It is necessary to customize the creation of this object based on the browser being used by the user. The function handles creation for Internet Explorer, Firefox, Safari, and Opera.

Hold Search Functionality:

This set of functions allows the user to set parameters to restrict the hand history files retrieved based on the hold cards the user is dealt in the hand. These files allow the user to select and deselect various card combinations from a table by altering the table's cell background color.

void [card category to swap] (integer toggle, boolean suited)

The two cards dealt to a user in the beginning can be put into one of the following categories: a pair; a non pair, suited or unsuited, a connector (cards with adjacent values, i.e 7,8 or Jack, Queen), suited or unsuited, a one-gap (cards with separated by two values, i.e 2,4 or Jack, King), suited or unsuited. Based on the set of cards a user would like to select or deselect, the appropriate function is called. If the user wants to select the cards, *toggle* is set to 1, 0 for deselect, and *suited* is set to 1 if the user is referring to the subset of cards in the category which have the same suit, or 0 to denote the fact that they are referring to the unsuited variety (obviously always 0 for pairs). Based on these two parameters, the function loops through the table of cards and changes the background color as needed.

View History Functionality:

This functionality allows the user to view and hide the hand history for a particular hand returned after executing a search.

void viewHistory(integer toggle, integer handid)

The *toggle* parameter selects whether or not a user wishes to show or hide the hand history. If the user wishes to hide the history, the container holding the history text is cleared, if the user wishes to view the data, an appropriate call is made to the gethistory.php function which returns

the hand history data corresponding to the id stored in *handid* for the user to view. Also printed is a link which allows the user to hide the information.

3.4 PHP Logic Layer

search.php

This rather simple file generates html to display in the interface allowing the user to set search parameters. The different search parameter sections are listed as functions and are called based on an *opt* parameter which is passed to the file though the an HTML POST parameter. The function examines this parameter and then calls the appropriate function based on the value. Each individual function performs the same general task of outputting some HTML text which allows the user to select various criteria to search on. It is necessary to generate the HTML dynamically since for options such as opponents, the data needs to take into consideration the current user and populate drop down menus and check boxes accordingly.

statistics.php

This file functions as a hub, taking in requests for hand histories and returning the identification numbers of hand histories which match the criteria. The function must first take in any POST variables passed to the file and build an SQL statement from the data. This statement is then executed and the resulting IDs found are output to the screen with a check box, identification information, and a link to provide the user the option of viewing the hand history in their browser via an AJAX request to gethistory.php. Over all, this function is rather simple as well and mainly provides a bridge between the search functionality and the more complicated logic found in export.php, gethistory.php, and generatestats.php.

The specific method of accomplishing this is as follows:

```
 // Build the Query

If Parameters are passed to the server

    Build a custom SQL statement based on parameters

Else

    Use stock SQL statement to get all hand history files

End If

// Get the results

Connect to the database

Execute the query

// Create HTML
```

```
    Loop through results

        Output a table row which displays the game number and checkbox.

        Output Link to display Hand history

    End Loop

    Provide Link to download hand histories of selected games.
```

import.php

This is called whenever the user wishes to import a hand history file into the system's database. After the file has been uploaded to the server, its contents are parsed and inserted into the database. The only parameter the function takes is the data file, which is deleted when the parsing is complete. The general strategy of the parser is to examine each line of the text file and search for a key word based on the line's structure. Once the line has been interpreted, a SQL statement is formed and the information is stored in the database. While the structure of this file is simple, there is a great amount of logic that needed to be developed to handle this functionality.

export.php

When the user wishes to export hand history data to a text file for downloading, this function is called. The function takes in a list of hand history IDs, which it loops through calling gethistory.php each time and passing the hand history ID and a file name which is then used by gethistory.php to write the hand history to the file. When this finishes, the function outputs a link to where the user can download the text file to their computer.

gethistory.php

This function deals with providing a text output, either to a file in text form or to the screen in HTML form, of a hand history. The function takes in a hand history ID, and a file name, which can be null. If the file is null, the output is in HTML form to the screen, if it is a legitimate file destination, the output is directed to the file and a link to the file on the server is presented to the user for download. The basic structure of this function is as follows, first the function uses sp_history.php to generate an SQL function to retrieve from the database the list of actions for the particular hand history. This list is then interpreted and put into user friendly phrases which are then output to the appropriate destination.

generatestats.php, etc.

This is no more than a collection of functions which take in a list of hand history IDs and a set of codes which identify a list of statistics to be generated based on the list of hand histories. Each function is basically the same, consisting of a call to the database to recover the raw information needed to generate the statistic, followed by the logic necessary to generate the

statistic information from that data. Lastly, HTML is dynamically generated to display the results to the user either in graphical or tabular form.

3.5 Configurations and Data Layer

sp_history.php, etc.

There is no functional code in this file, but it does contain the SQL used to recover the list of hand actions used to display a hand history file to the user. Likewise, there are several other files such as this which are used to recover the data used to generate the statistics reports. These files are used by the *gethistory.php* and *getstats.php* files.

config.php

There is no functional code in this file either. This file contains configuration information for the system, such as database passwords, table names, and other server parameters referenced by all of the files in the PHP layer.