Website Analysis Data Display

Alan Wu

Version History

2006.12.22 - Version 1.0 - Initial document.

Contents

1 Introduction

One of Gamesman's defining characteristics is its ability to perfectly solve games by exploring all possible positions. By walking the complete game graph, Gamesman can gather all kinds of data that can then be analyzed to produce all sorts of information, from simple position value summaries to interestingness analyses. However, this data is most useful if presented from a single, central location in a human-readable format. The Gamescrafters website, http://gamescrafters.berkeley.edu, thus has a section devoted to the display of analysis data. This document describes how to generate the necessary data files for use on the website, as well as attempting to describe how the analysis data display backend operates.

2 History of Analysis display

The original version of the website's analysis section was a PHP script, written by Robert Liao, that, when given a game name and a properly formatted template, would retrieve the XML file containing the game's analysis data, and display it to visitors according to the format of the template.

The current analysis team, consisting of Alan Wu and Matt Johnson, assumed responsibility for this script, and updated it to display the new data Gamesman was now capable of generating (detailed position values for each remoteness level of a variant). This version then served as a template for the current iteration of the website analysis display, which was made as dynamic as possible in order to simplify data updates.

3 Components of Analysis display

In general, the Analysis section of the website consists of five discrete components: an overall listing for each analyzed game, showing summaries for all of the game's variants; a detailed summary for each game's variant, showing position statistics for each remoteness; an overall listing of all analyzed games, showing the summary for each game's default variant; and a listing of the status of each game. Henceforth, these will be known respectively as the "Summary of default variants", "Summary of game variants", "Detailed variant information", "Game status", and a "Navigation frame" linking to each analyzed game's page as well as the game status page.

3.1 Directory structure

While a complete overview of the directory structure of the Gamescrafters website is out of the scope of this document, it is necessary to give the reader a view of the structure of the relevant analysis directories, and the location of important files therein. The directory and file structure is as follows:

```
|--analysis
|--|--archive (this directory contains various archived pages)
|--|--data (this directory contains the analysis data files)
|--|--|--allgamesinfo.txt (text file containing default variant for each
|  |  |  game, along with other misc. info)
|--|--|--allgamestatus.txt (text file containing game statuses)
|--|--|--m*.xml (data files for overall summaries of games)
```

```
|--|--|--m*\_\#.xml (data files for specific variants)
|--|--framework
|--|--|--framework.php (main header that forces include of template.php)
|--|--|--template.php (fills template files with data parsed from XML files)
|--|--images
|--|--templates (templates used for analysis data display)
|--|--analysisIndex.php (displays summary of default variants)
|--|--gameanalysis.php (displays summary of all variants of a game or
|  |  detailed information for a single variant)
|--|--gamestatus.php (displays game statuses)
|--|--nav.php (creates navigation frame)
```

All files not specifically mentioned are either obsolete and deprecated or not specifically relevant to understanding how analysis data is displayed.

3.2 Summary of game variants

This was the initial version created by Robert Liao, and is handled by the PHP script **analysis/gameanalysis.php**. For each game, it summarizes the value, position statistics, and some miscellaneous information for each variant. The information is retrieved from XML files located inside **analysis/data**. These XML data files are generated by Gamesman, either through the Analysis menu of the text interface, or by including the `--analyze` command line option. Gamesman places the generated analysis data inside **[Gamesman root]/bin/analysis/[game name]/xml**. For more details on generating analysis data, please refer to the Analysis documentation or Section 4.3 of this document.

The XML files are parsed, and the data contained therein is formatted and displayed according to the template file **analysis/templates/gameanalysis.tpl**.

3.3 Detailed variant information

This page is merely an extended version of the above section; the script that it uses is almost identical to the version created by Robert, with some modifications to allow it to use a different XML data file and template to display detailed variant information. The presence of the HTTP GET variable **variant** informs **analysis/gameanalysis.php** that the viewer wishes to see detailed variant information; the script then loads the XML file **analysis/data/[gamename]_[variant].xml**, parses it, and formats and displays the data using the template **analysis/templates/gamevaranalysis.tpl**.

3.4 Summary of default variants

This page is the main analysis page, the first page visitors see, and was originally a large amount of text extolling the (somewhat questionable) virtues of game analysis; it was wisely decided that visitors would be more interested in seeing *real data* than reading a dry piece of text written late at night. It now displays information about the default variant of each game, in a format similar to each game's summary page.

This section is handled by **analysis/analysisIndex.php** implemented in a rather (in the author's mind) clever manner – include **analysis/gameanalysis.php**, have it do the heavy work of XML parsing, then simply grab and display only the data for each default variant, ignoring the rest.

The default variant of each game is retrieved from the file **analysis/data/allgamesinfo.txt**, the generation of which is discussed in Section 4.3.

3.5 Game status

This page lists the status of all games implemented in Gamesman. This includes the names of the C and Tcl (if any) authors, the level of the GUI (Text, Bronze, Silver, Gold), status of debug flags (kDebugMenu and kDebugDetermineValue), and miscellaneous information (at the moment, this consists of which help strings have yet to be written). The information displayed on this page is gathered from the text files **analysis/data/allgamesinfo.txt** and **analysis/data/allgamestatus.txt**.

Originally, this page was static and was sorely outdated (it could only be updated by hand). However, it is now dynamically generated by **analysis/gamestatus.php**, and updating is simply a matter of running a script and placing the generated text files in the proper location.

3.6 Navigation frame

This page appears in the left frame of every analysis page; it contains links to the summary pages of each analyzed game, as well as a link to the game status page. It should be noted that the list of analyzed games displayed by this page is not necessarily identical to the list of analyzed games shown on the default variants page (3.4) (the former is a superset of the latter). This is because some games defined their variants in such a way that the hash code for the default variant is larger than the maximum number of variants; when automatically generating analysis files, it is impossible to determine if this is the case, and so for some analyzed games, there is no data for their default variant. Thus, the default variant summary page only lists games with data files for their default variant, while the navigation frame lists all games that have at least the data file summarising all variants.

Like the game status page, the navigation frame was at one time static, but it is now dynamically generated by **analysis/nav.php**. The PHP script reads **analysis/data/allgamesinfo.txt** to create a list of all games, and then removes from the list any games that do not have a **analysis/data/[gamename].xml** file.

4 Data files used by Analysis display

There are two types of data files used: XML data files, and text files. These data files are used by the various PHP scripts to dynamically generate the analysis pages.

4.1 XML files

The XML files are located in **[gamesman root]/bin/analysis/[gamename]/xml**. There are two types of XML files created, **[gamename].xml** and **[gamename_[variant].xml**. The former contains overall data for each variant, while the latter, one for each analyzed variant, contains detailed information for the variant. The format of the files, in keeping with XML's goal of being both computer and human-readable, is rather self-explanatory.

The XML files are generated by the functions **writeXML()**, **writeXMLData()**, and **writeXMLVarData()**, found in **src/core/analysis.c**.

4.2 Text files

There are two text files used, both located in **analysis/data/** in the Gamescrafters website directory structure. These are **allgamesinfo.txt** and **allgamestatus.txt**.

4.2.1 allgamesinfo.txt

This file contains a number of records, one for each game. A sample record is shown below:

```
IceΩBlocks^iceblocks^19^KevinΩDuncanΩandΩNeilΩTrotter^False^False^GUIΩhelpΩnotΩwritten
```

There are a total of 7 fields, with each field separated by a ^; spaces are replaced with Ω. The reason for this strange choice of delimiters is that some games can contain commas in their names, while others have multiple words, so it is necessary to choose delimiters that won't be used (Alan says he'll cry if either of the chosen delimiters actually end up in game/author names). The fields are as follows:

1. Long game name
2. Short game name
3. Default variant
4. C Author
5. Debug menu
6. Debug determine value
7. Misc. info (currently unwritten help strings)

4.2.2 allgamestatus.txt

This file contains a number of records, one for each game with a GUI. Similarly to **allgamestatus.txt**, fields are separated by a ^, while spaces within records are replaced with Ω. A sample record is shown below:

```
Connect-4^Gold^EudeanΩSun
```

There are a total of three fields, listed below:

1. Game name
2. GUI status
3. Tcl author

4.3 Generating the data files

Before generating the various data files, ensure that your copy of Gamesman is up-to-date. Perform the following commands inside the Gamesman root directory:

```
cvs update
make clean
make
```

4.4 Generating the XML files

Perform the following commands:

```
cd bin
./newsolve analyze
```

newsolve is a shell script, written by John Jordan, that will automatically solve all games. The **analyze** option tells it to invoke each game with the command line options: `--solve all --analyze`. This tells Gamesman to generate XML files for each game. These XMl files can be found in **bin/analysis/[gamename]/xml**. These XML files should be placed in **analysis/data/** in the Gamescrafters website directory structure.

4.5 Generating the text files

To generate the text files **allgamesinfo.txt** and **allgamestatus.txt**, perform the following commands:

```
cd bin
./allgamesinfo
```

The generated files are located in **analysis/**; these should be placed in **analysis/data/** in the Gamescrafters website directory structure.

5 Analysis display backend

The backend for analysis data display (the components of which were described in the previous section) is written in PHP. The original code was written by Robert Liao, and, for the most part, survives intact (with the addition of a few ugly hacks). It should be noted that due to unfamiliarity with the PHP language, the modifications made are not very pretty and could almost certainly be made more efficient (hence why they're called "hacks"). It should also be noted that, due to the aforementioned unfamiliarity, as well as only a partial understanding of Robert's code, the following description of the backend's inner workings may, in turn, be difficult to understand.

Very generally, the backend consists of a PHP script that includes **framework/framework.php** (which includes **framework/template.php**) and parses an XML data file. The parsed data is then used by **framework/template.php** to fill a template (**templates/*.tpl**), which defines how the data is to be displayed in the user's browser.

5.1 Template format

Templates, located in **templates/** are HTML files with a specially defined **{BLOCK *blockname*}{/BLOCK}** sections. **BLOCK** sections cannot be nested, and no more than one **BLOCK** may be defined with the same name (this is due to limitations in **framework/template.php**.

A block can be thought of as a "hierarchy level tag" in the XML data file that (usuallly) repeats, for example **variant** in the overall game XML files, (**data/[gamename].xml**, or **remoteness** in the variant-specific XML files, **data/[gamename]_[variant].xml**. In the template files, each set of block tags surrounds a section of HTML code that (in almost all cases) defines the appearance of a single row in a table. This is why blocks correlate to tags in the XML file that

repeat: the backend script creates a number of rows equal to the number of times an associated tag repeats, and fills and displays each row with the data contained within a single instance of that tag.

Each block must be named according to the name of the associated tag; for example, a block intended to display data associated with the **remoteness** tag should be named **remoteness**, like so:

```
{BLOCK remoteness}
Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.
{/BLOCK}
```

To specify specific data items that should be displayed, type the tag hierarchy (starting with the tag corresponding to the block), with a period, ".", delimiting each tag, and surrounded by braces ({}). So, continuing with the above example, a completed block section might look like this:

```
{BLOCK remoteness}
<tr>
<td>{remoteness.level}</td>
<td>{remoteness.win}</td>
<td>{remoteness.lose}</td>
<td>{remoteness.tie}</td>
<td>{remoteness.draw}</td>
<td>{remoteness.total}</td>
</tr>
{/BLOCK remoteness}
```

Variables defined and set by the backend script may also be used; simply type the name of the variable, enclosed by quotes, and the script will replace it with the value of the corresponding variable. See **templates/gamevaranalysis.tpl** for a good example of the various things that can be done with a template file (this is the template file responsible for displaying detailed position value information for a specific game variant).

5.1.1 List of template files

These template files can be used to assist in the creation of a new template:

- **templates/gameanalysis.tpl**, template for overall summary of each variant of a game
- **templates/gameanalysismain.tpl**, template for overall summary of default variant of each game
- **templates/gamevaranalysis.tpl**, template for detailed summary of specific variant of a game

5.2 Backend scripts

This section describes the operation of each of the backend scripts, and how to create a new one (it is highly suggested that an existing script be used as a sort of template for the new script). **framework/template.php** is described last, as it is the least understood script (despite its importance in the overall scheme).

5.2.1 gameanalysis.php

This is the original script by Robert, as well as some modifications made to allow it to handle different XML/template files depending on the presence of the HTTP GET variable **variant**. It is responsible for displaying overall summaries for each variant of a game and detailed summaries of specific variants of a game. In general, it is recommended that this file be used as the basis for any new scripts that wish to perform a similar task.

The bulk of the script consists of the element handlers required for XML parsing; these handlers are also responsible for filling the (confusingly named) template object's internal arrays, from which the data to fill the appropriate template file is taken. These element handlers are:

- `ANALYSIS_start_element($parser, $tag, $attributes)`
- `ANALYSIS_end_element($parser, $tag)`
- `ANALYSIS_char_data($parser,$cdata)`

The function `ANALYSIS_is_template_tag($tag)` is a helper function used by the element handlers, while `ANALYSIS_fillTemplate($filename)` sets up the XML parser, parses the XML file, and fills the template object).

The rest of the script is not enclosed in any function; this causes it to be automatically called when the script is run. The important things to note are the following lines:

- `$template = new FrameworkTemplateFile([string containing path to template file to use]);` (creates a new template object that will use the specified template file)
- `ANALYSIS_fillTemplate([string containing path to XML file to use]);` (call to function which will fill template object with data from specified XML file)
- `$template->display();` (call to method that will display filled template)

The rest of the code is template object initialization or error checking.

One final note: The following lines in `ANALYSIS_fillTemplate()` are of vital importance:

```
foreach($ANALYSIS_frameworkData_variant as $key => $frameworkData)
{
$template->ArrayFillBlock("variant",$frameworkData);
}

if(array_key_exists("variant",$_GET))
{
foreach($ANALYSIS_frameworkData_variant_detailed as $key => $frameworkData)
{
$template->ArrayFillBlock("remoteness",$frameworkData);
}
}
```

Both `foreach()` loops are functionally identical (the second runs only if "variant" is present in the HTTP GET array), so only the first will be discussed. The `foreach()` loop takes the array $ANALYSIS_frameworkData_variant, which was filled with data contained within the children tags of the XML tag "variant", and places its contents into the block "variant". The contents of this block are then used to replace the BLOCK variant/BLOCK section in the .tpl file. All scripts that will process XML files using the template framework require a loop similar to this one (modified appropriately) or they will not function properly.

5.2.2 analysisIndex.php

This script is responsible for displaying the summary for the default variant of each game; this is seen when visitors view the main analysis page. This is accomplished by loading **gameanalysis.php** as a library, and using it to parse each game's overall summary XML data; this results in an array `$defaultVariantInfo`, each index of which contains the parsed data for each variant. The data contained in the index corresponding to the default variant is then placed in the template object. After the data for each analyzed game is placed in the template object, it is formatted and displayed according to the template file **templates/gameanalysismain.tpl**.

5.2.3 gamestatus.php

This script is responsible for displaying the status of each game. As of the writing of this document, this status consists of the game name, GUI status (Text, Bronze, Silver, or Gold), analysis status, whether kDebugMenu or kDebugDetermineValue are set to true, any help strings that have not yet been written, and C and Tcl (if any) authors.

This script uses data found in two text files, described in section 4.2. The script itself simply iterates through each line in **data/allgamesinfo.txt**, cross-references with the data in **data/allgamestatus.txt**, and creates a row in the table for each game found.

5.2.4 nav.php

This script is responsible for displaying the navigation frame in the left frame of the analysis pages. It works similarly to the game status script; it iterates through **data/allgameinfo.txt** to discover analyzed games, and then creates links to each game's overall summary page. Analyzed games are considered to be those with an overall summary XML data file, **data/[gamename].xml**.

5.2.5 framework/framework.php

A rather simple script; it is included by any script that wishes to use the template framework, via the following lines:

```
$FRAMEWORK_LOAD=TRUE;
require_once("framework/framework.php");
```

It simply loads the file **framework/template.php**, which fills and displays template files with the proper data.

5.2.6 framework/template.php

This script is somewhat arcane; it defines the template object class that is used by scripts that include the template framework. The author of this document has commented the various functions to the best of his ability, and hopes that they are sufficient to describe the purpose of each function. Should the reader wish to gain deeper understanding, it is recommended that he either speak with Robert Liao (the original author of the template framework) or follow the code path, beginning with the script that includes the template framework (e.g., gameanalysis.php), and keep track of the internal data structures with pen and paper. No modifications of this file, however, were required during the course of extending the analysis display backend.

6 Miscellaneous

6.1 Page caching

It is a well-known fact that the OCF servers' PHP parsing is slow; since analysis data does not change very frequently, it therefore makes sense to cache generated pages in order to speed up page accesses. Thus, if the modification time of the cached page (if it even exists) is earlier than the modification time of the appropriate XML data files, the new generated page will be cached and stored for later accesses; if the cache is newer than the data files, then the cache is returned, instead of reparsing the XML data files.

The page caching is implemented using PHP's output buffering; all output is sent to a buffer, written to the cache, and then flushed to the browsesr. There is a small amount of logic involved: if the cache exists, and is younger than the modification time of the **data/** directory, or the PHP script, then the cache is displayed; otherwise, generate a fresh version, cache it, and display the fresh version. Unfortunately, for some reason probably attributable to the OCF server, the performance improvement is somewhat ... *variable*. In theory, however, it is better to have the caching than to not have it at all.

6.2 Table sorting

All tables can be sorted by clicking on a column header. Sorting is accomplished by the Javascript library **Sorttable**, written by Stuart Langridge and released on his website under an MIT-like license. Details on the implementation of this library can be found on the author's website.

6.2.1 Using the Sorttable library

Using the Sorttable library is very simple. First, ensure that the HTML document includes the Sorttable library, by including the following line:

```
<script type="text/javascript" src="sorttable.js"></script>
```

For each table that is to be made sortable, give it a unique **id** attribute, and make it of class **sortable**. An example is shown below:

```
<table cellspacing="0" class="sortable" id="summary_table">
```

There is one final detail. Some rows (e.g., rows containing "totals") should not be sorted. In that case, make the row(s) that should not be sortable of class **sortbottom**. An example is shown below:

```
<tr class="sortbottom">
```

That is all the information that is needed in order to make a table sortable.

6.2.2 Changes made to the Sorttable library

A few minor changes were made to the Sorttable library so it could properly handle some idosyncracies of the analysis tables. First, maximum remoteness is displayed as **Inf**; obviously, we want this to be sorted just like any other number (i.e. ∞), so the function `ts_resortTable()` was modified to call the numeric sort function when **Inf** was encountered, instead of calling the alphanumeric sort. `ts_sort_numeric()` was modified to return the proper value (-1 or 1, depending on order of arguments) if **Inf** was the value in the cell to be sorted.

A second modification was made to `ts_resortTable()` and `ts_sort_numeric()` to properly handle numbers followed by a percent (%) sign. This is due to the "Hash Efficiency" column displaying percentages; we want this column to be sorted numerically, rather than alphanumerically. A change similar to the one made to handle **Inf** was made so that `ts_sort_numeric()` would be used to sort cells with values of the form **/d.+%/**. `ts_sort_numeric()` was then modified so that it would properly handle the aforementioned cell value type; the "%" is removed, and the remainder (simply a number) is then sorted as usual.