ShadyStats: Visualizing Game Statistics using Hierarchical Parallel Coordinates

Mike Vlad Cora
University of British Columbia
CS 533C: Information Visualization

Abstract

In this time of blazing fast graphics hardware, surround sound processors, and oversaturated gaming industry, a video game's AI and gameplay are playing larger and larger roles setting it apart from the rest. Statistical analysis is almost a given when designing a tuneable AI system. Statistics should be gathered throughout the development cycle: both during experimentation with different algorithms and gameplay mechanisms, and during feature development and final bug fixing. The ShadyStats is introduced, utilizing hierarchical parallel coordinates to visualize a game's multi-dimensional statistics.

Additional Keywords: multi-dimensional statistics, parallel coordinates, dynamic queries, video games.

1 Introduction

In this time of blazing fast graphics hardware, surround sound processors, and oversaturated gaming industry, a video game's AI and gameplay are playing larger and larger roles setting it apart from the rest.

Probability and randomness are the two pillars of an interesting video game AI, which does not become repetitive, predictable and dull. In this kind of approach to game design, the probability knobs become the de-facto tuneable values used to guide the game play experience. The upside of this approach is that it delivers often surprising, unpredictable behaviour that has long replayability value, and feels more human. This is a bold statement, but since most human actions are influenced by so many different factors, one can say we're basically random, with various skill and probability knobs adjusted. A human would never play a game exactly the same every time, and neither should the AI.

The flip side of probability and randomness is how to measure that in general, the AI is behaving how the designers intended. This is hard since the variety of behaviour has just exploded by adding a mere few dice rolls in the mix. Such AI has to be evaluated by gathering a lot of statistics over time, and analyzing the trend lines, outliers, and relationships between different dimensions. They should be gathered throughout the lifecycle of the game: while experimenting with different algorithms, prototyping different gameplay features, and during actual development and final bug fixing. An info visualization tool to facilitate the interpretation of these statistics can only have a positive effect on the final quality of the game.

1.1 Dataset

The dataset used was generated from an over-the-top arcade soccer video game, where the rules of soccer are greatly relaxed. It includes statistics from about 2400, 5 minute, AI vs. AI games played at six different dates over the span of the last 2 months of development, consisting of 15 dimensions (ie. shots, goals, passes), times 2 for Home/Away. That would make 2400 games * 15 * 2 = 72,000 numbers, a pretty modest dataset, but still large enough to overwhelm an Excel spreadsheet.

Having designed and implemented the AI system which generated the dataset during this past year, I have a fair bit of expertise with the domain, however, I have never done any sort of statistical analysis (and even less multidimensional analysis).

1.2 Tasks

The ShadyStats addresses the following overall goals, and is intended for a variety of professional roles involved in the production of a video game. It aids in the statistical analysis of multi-dimensional data gathered through the game's development cycle, with the following goals in mind:

- Clearly show correlations between different metrics (ie. # of shots should hopefully be positively correlated to # of goals).

- Help detect bugs by highlighting outliers and abnormal differences between selected dimensions. For example there may exist subtle bugs in the formation and positioning code and/or tuning parameters that can lead to differences between the Home and Away teams, giving one an unfair advantage over the other. A lot of times, these differences are not caught by play-testers, or their causes are hard to pinpoint.

- Help with tuning a balanced progression through difficulty levels by mapping it to the different dimensions. AI skill tuning continues to be a difficult challenge that a large number of games fail on, especially since tuning and balancing is often left to the very end of a mad, deadline driven development cycle (not always though). Anything that would hasten this process would improve the quality of the final product.

- Mapping the data to the time it was recorded should provide an interesting visual history of the AI implementation and tuning. In addition, bugs and undesired behavior would be easy to spot as soon as they are introduced, with a "base-line locking" feature: the interface would allow the user to save a snapshot of good statistics, against which future data samples can be compared, highlighting any differences.

- Aid communication with the publisher throughout the lifecycle of the project. This is a more nebulous goal of presenting an alternate method for information flow, especially across language barriers.

2 RELATED WORK

The information visualization technique that I thought would be suitable for addressing the tasks above was hierarchical parallel coordinates. Although confusing at first, parallel coordinates[1] can be an effective way of quickly displaying a large number of records across many dimensions. There are two main downsides to using parallel coordinates, the first one being that displaying a line for each record across all dimensions can lead to a lot of clutter, fast. And the second being that dimensions closer together on the display are the easiest to compare, with relations between dimensions far apart having little chance of being discovered.

The XmdvTool[7] hierarchical parallel coordinates[2] C++ implementation addresses the first problem very nicely, by aggregating the data into cluster trees, then displaying a shaded band of colour representing the extents, around the solid mean line. It implements a variety of other visualization and navigation techniques in addition to parallel coordinates. For the purposes of the ShadyStats, I only used the hierarchical parallel coordinate component, as described in section 3.2.

In order to somewhat address the second problem of parallel coordinates, a second visualization system, namely a scatter plot is used to augment and get more detailed information than the aggregated clusters. I expect the clusters to be most useful most of the time, and zooming in all the way to the individual record level to only be useful for very sparse queries, or for investigating outliers. The graphing component used is ZedGraph[8], and is described further in section 3.3.

Another tool I wanted to evaluate was the ILOG Discovery Preview[11], but was unable to get their free license to run on my computer. It is implemented in Java, and seems to contain a lot more features than I would need, so Xmdv seemed like (and was) the right choice.

The dynamic queries, filtering, data linking and zooming techniques used in the ShadyStats have been well researched and covered in the information visualization field [5][6]. My contribution is more to the gaming industry, attempting an academic approach to developing fun. In my three and a half years in the video gaming industry, I have yet to see statistics visualization being involved in the development cycle of a video game. This may be due to the games I have worked on (car racing and soccer). I imagine that the teams in charge of real time strategy, or simulation games (ie. SimCity) are probably knee deep in statistical analysis. However, a visual analysis tool would be beneficial for any type of game, since everything that can be measured has the potential to be useful, when presented correctly.

3 SHADYSTATS

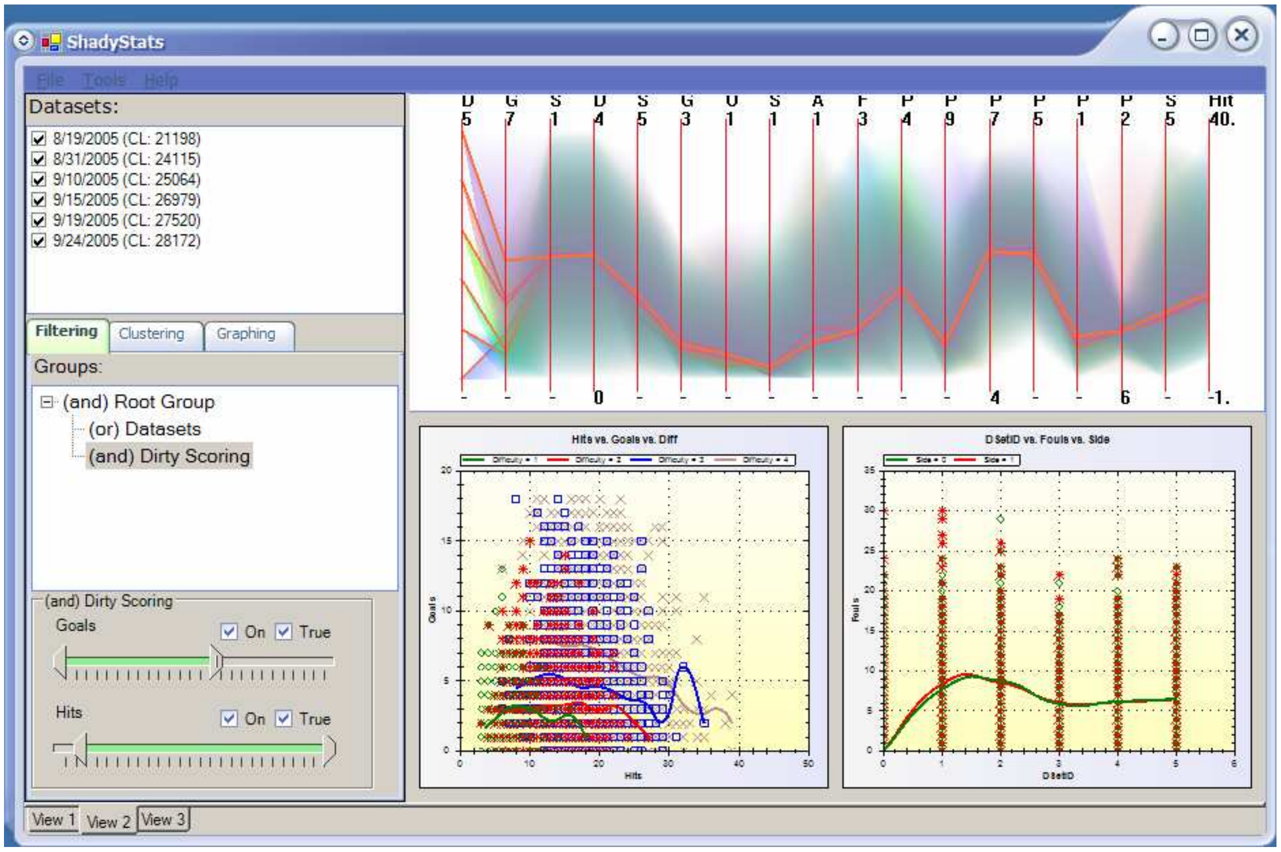
The ShadyStats is meant as a highly interactive, easy to use, yet very informative live exploratory tool that gives feedback after every update made to the viewing parameters. All different components are linked live and update depending on the clustering, filtering, and other view options modified. All view parameters can be saved in different "View Canvases", which can be switched at any time.

3.1 Features Implemented

The following features have been implemented and will be described in more detail:

- Maintains history of datasets for easy comparison throughout the development lifecycle.
- Uses the hierarchical parallel coordinate implementation in XmdvTool[7] for high-level trend overviews.

Figure 1: The ShadyStats, with a filter on number of goals and number of hits, displaying 6 datasets from different dates.



- Generate detailed scatter plot graphs on demand, using ZedGraph[8], a very well written and documented C# charting library. I highly recommend it.
- In depth, fully controllable clustering and filtering of the information.
- Save all filtering, clustering, graphing and view settings into different "View Canvases", for fast reproduction of visualizations.

The solution is implemented in Visual Studio.Net, for several reasons:

- XmdvTool is a C++ application.
- ZedGraph is a C# library.
- C# and C++ play very nicely together.
- C# is very UI friendly, and has a fast development and turnaround time, without getting bogged down in details not pertaining to implementing the solution.
- I have less (and outdated) experience with Java.

3.2 Hierarchical Parallel Coordinates

Hierarchical parallel coordinates was chosen as the information visualization technique to communicate the large amount of multi-dimensional statistics. This technique is described in [2] and implemented in XmdvTool[7], which is a stand-alone C++ Exe demonstrating various other multi-dimensional visualization techniques.

Since too many people seem to like Microsoft bashing, I feel that I must take a bit of space to commend Microsoft on a job well done with the .Net framework, and Managed C++. Exposing the hierarchical parallel component implemented in XmdvTool to C# was a breeze, with the development environment giving me all the tips needed to do the job. I did not even have to consult the Google oracle, nor the Visual Studio Help. I have had fairly extensive experience with C++ and C# individually, but have never brought the two together, and I never imagined how seamless the interoperability would be.

The lengthiest task was deciphering the structure of XmdvTool, and isolating the relevant pieces. It was written using Tcl/Tk[6] user interface scripting system, so the tool was relatively well separated architecturally between a GUI layer and a back-end layer, however it makes use of a lot of global data pointers which become a problem when exporting functionality intended to be encapsulated and self-contained. This remains a problem, and an application can only have one Xmdv parallel coordinate control per application, since it makes use of global data pointers, and some nasty stomping issues can occur otherwise. Future work would be encapsulating all the currently global data pointers into the Managed C++ control, so multiple instances are possible.

3.2.1 Clustering

The automatic clustering performed by XmdvTool was not well suited for this application, since it did not take into account pre-existing domain knowledge. There are specific dimensions in this case that make sense to cluster under, namely the dataset, difficulty level, and side being the top three. Automatic principle component analysis ignores this, and creates "meaningless" clusters that would not help with the tasks described in section 1.2.

The clustering is simple hierarchical "binning" based on the dimension corresponding to the given tree level. For example clustering by dataset would create six clusters by aggregating the min, max and means of all the child clusters, with the individual games being the bottom-most single entry, leaf clusters with min=max=mean, and no children. Any number of dimensions can be added to the clustering tree hierarchy (assuming infinite RAM and processing power), however only a few make the most sense, namely dataset, difficulty and side. Other dimensions could be added through the interface, and used for example to determine what makes high-scoring games different from low scoring games, by adding goals to the cluster list.

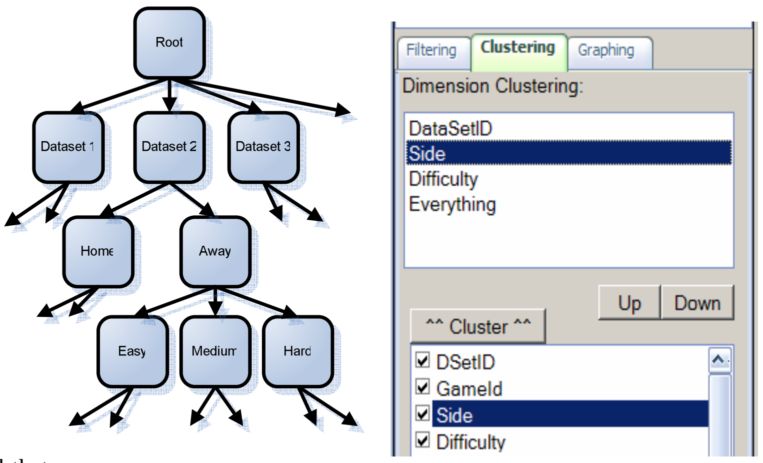


Figure 2: The cluster tree (left) corresponding to the settings on the right.

I also found the brushing tree navigation technique implemented in XmdvTool difficult to use for the specific tasks mentioned. I can see how it is useful when exploring for relationships in an unknown dataset, however it was an unintuitive exploration technique for viewing details from explicit, known data viewing "angles". The simplified solution is to simply select the tree level that corresponds to the level of detail desired. For example selecting Side above would show (2 * num datasets) number of mean lines, since the Side is a child of Dataset.

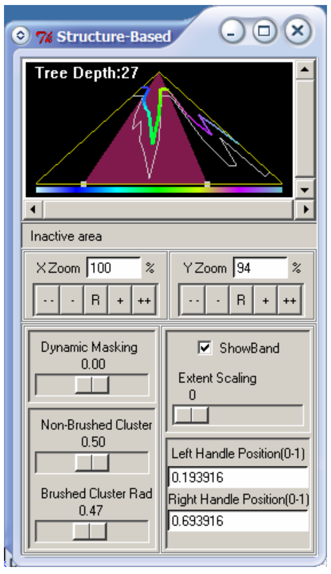


Figure 3: Xmdv Structure-Based Brushing

3.2.2 Filtering

XmdvTool contains a brushing feature for the flat parallel coordinates component, but this did not function in the hierarchical shady component. More investigation is needed into the exact implementation of Xmdv's brushing mechanism. In the mean time, a simple query-based mechanism was implemented, filtering on ranges of different dimensions. This feature is key to exploring different aspects of the dataset (for example comparing how the different team difficulty levels play against each other). A logic tree-based filtering system was implemented that allows any possible formal logic filtering statement to be specified through the GUI.

This solution is a bit clunky and takes up a lot of unnecessary space. Different techniques should be investigated of merging the different attributes into less intrusive mechanisms than the check boxes for Active and Positive/Negative, making it possible to display more filters at once. Also note the lack of labels, this will defiantly need to be addressed.

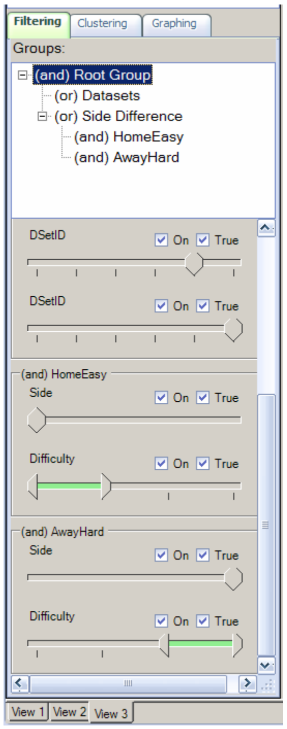


Figure 4: Visual filter specifying the following query:
(Dataset=5 OR Dataset=6) AND
((Side=Home AND Difficulty=Easy To Medium) OR
(Side=Away AND Difficulty=Hard To VeryHard))

It consists of filter Group nodes that can either OR all of their child nodes, or AND all of them (see Figure 4). They can be individually turned on or off, set to positive or negative (not logical operator), and be recursively nested without limit. The individual Filters are children of the Group nodes, and apply to one dimension only, specifying the Range of inclusion. They can also be individually turned on or off, and set to positive or negative. The range control is pretty common in the wild, but that may not have always been the case. One noteable implementation, famous in the infovis world is in the FilmFinder[5]. The particular implementation used in ShadyStats is called ZzzzRangeBar[9].

3.3 Scatterplot Graphs

The ZedGraph[8] control written in C# was used to draw the detailed scatter plot graphs, with an average mean line drawn through the data. It proved to be very well implemented, full of features, and superbly documented. It has built-in zooming, panning, and even CopyToClipboard and SaveToFile functionality, so pictures for presentations and documents can be very easily generated.

A third dimension can be shown by binning the data into the extra dimension and displaying multiple curves, one per bin (ie. displaying a curve per difficulty level), as shown in Figure 5. Any number of graphs can be created per view canvas (limited by your memory obviously), and a very simple and naïve layout is applied, that can magnify a particular graph by squishing the other rows and columns. More relevant magnifying techniques are described in [6]. The Piccolo.NET[10] toolkit may prove useful for this application, and will probably be explored as a layout solution.

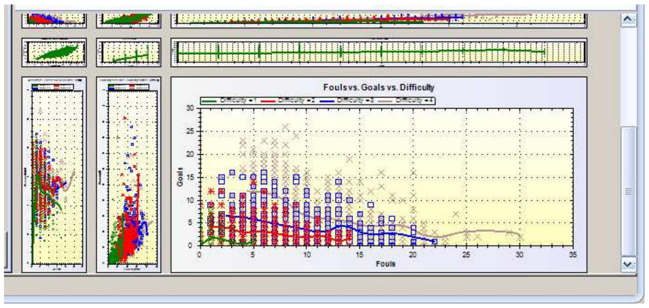


Figure 5: ZedGraph scatter plot, with 4 difficulty curves.

3.4 View Canvases

Please notice that at the bottom-left of Figure 1 there are three View tabs. All settings pertaining to a view (filters, clusters, graphs, zoom levels), are saved in a dataview structure, and can be restored at any time. These views can be saved to a file, and can be managed with source control, and quoted in bug reports during development. For example QA may have noticed an anomalous set of outliers. They can save the particular view canvas and send it to the programmer responsible, for further analysis (the scenario is described further in section 4.3).

3.5 Other Features

One can hide and re-order dimensions in the parallel coordinates component, and also zoom and pan it. This does not however seem to be particularly useful the way it is implemented at the moment, so it needs a bit more work.

The views can be saved, and a simple undo/redo system will be implemented that saves a copy of the dataview structure after each change to the filters, clusters, graphs or other view parameters. This structure is relatively small, and a history can be easily maintained.

4 SCENARIOS

The ShadyStats targets a number of professionals involved in the production of a video game, and is intended as a communication aid between people with different technical backgrounds, and roles. First, it will be the AI programmer's second stop when verifying that code does what is intended (first stop being the "breakpoint" in their respective development environment). The game designer will use it to measure and balance different game design features and difficulty settings. It can also become part of QA's regression cycle to make sure that the game is not broken or unbalanced inadvertently while fixing other bugs. And finally, it can be used by producers and managers as a communication aid with publishers.

All of the following scenarios are semi-fictitious, and would have been fully applicable if the ShadyStats was available earlier. They were created based on analysis of the real datasets (with 20-20 hindsight of course). The ShadyStats can be tailored and integrated into the development cycle to facilitate these kinds of discoveries early.

4.1 AI Programmer Scenario

The AI programmer has made some recent changes to formation positioning and passing code. A good baseline tuning has been established a week prior and stats were gathered. He wants to see if his recent changes could have introduced any imbalances when compared to the baseline.

The ShadyStats maintains a history of previous datasets, so the baseline is already in the list. The new dataset is loaded, and selected from the Datasets checklist along with the baseline dataset. Filtering and highlighting options are set for emphasizing differences between datasets (rather than comparing between different dimensions).

Things look alright at the lower difficulty levels; however the new passing code caused them to pass more times at the higher difficulty levels when compared to the baseline, apparent when aggregating the parallel coordinates based on difficulty level. Another apparent change is that the other team's pass intercepts shot up also, reducing the effectiveness of the offensive gameplay. The user selects the # of passes, and pass intercepts dimensions and creates a scatter plot to get a better view of the relationship.

The following pictures do not represent this particular scenario, but provide an example of an analogous scenario of comparing different datasets.

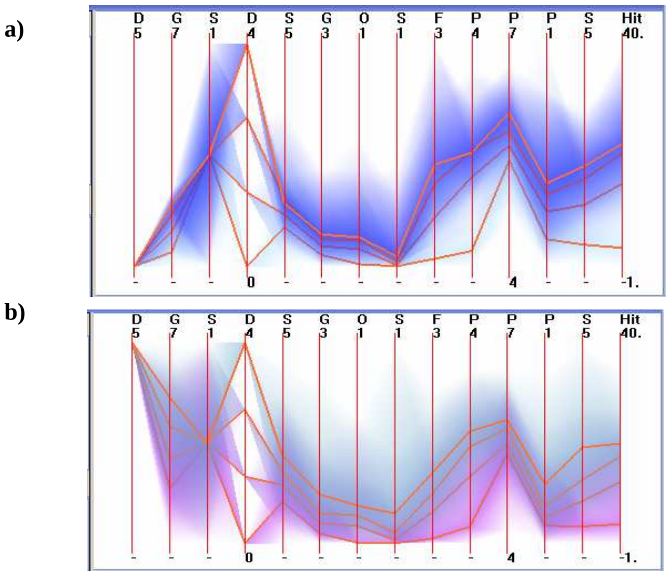


Figure 6: Difficulty level clusters: each line represents the mean of the cluster aggregating every game played at that difficulty level, and the shading highlights the distribution's mins and maxes. (a) is an example of unbalanced difficulty levels, and (b) is an example of a more balanced tuning.

4.2 Game Designer Scenario

Tackling a player to get the ball is so not soccer. This type of behavior has no place on the real playing field for good reason, and the rules discourage it. However this is what makes it all that much more fun, so it is a must in the game, but we cannot stop gameplay and award a red flag because it would interrupt the flow. Since taking an opponent out of play creates too much opportunity, there needs to be a penalizing factor, a detriment. In a well designed game, every such advantage should have a balancing disadvantage.

So the game designer decides to try awarding the other team a powerup that compensates them for being so rudely treated. The programmer said he hooked up the rule, but who knows with these programmers. It sort of feels right, but what does the ShadyStats say?

Plotting a graph of Fouls on the X-Axis, Goals on the Y-Axis, and grouping by Difficulty (resulting in four overlaid curves), clearly shows a negative correlation between Fouls and Goals (figure 5). So the more Fouls a team commits, the less likely they are to score, especially in higher difficulty levels, where tactics become more important than inelegant brute force. So the mechanism of awarding powerups to the offended team worked.

4.3 QA Scenario

There has been a decent baseline tuning established earlier, everything is practically bug free, but the darned AI programmer keeps wanting to "make it better". Well let's see if he broke anything. Loading the latest statistics that where gathered into ShadyStats and comparing it to the previous dataset should settle it. Aha, what's with the anomalous spikes, dumb programmer screwed it up again!

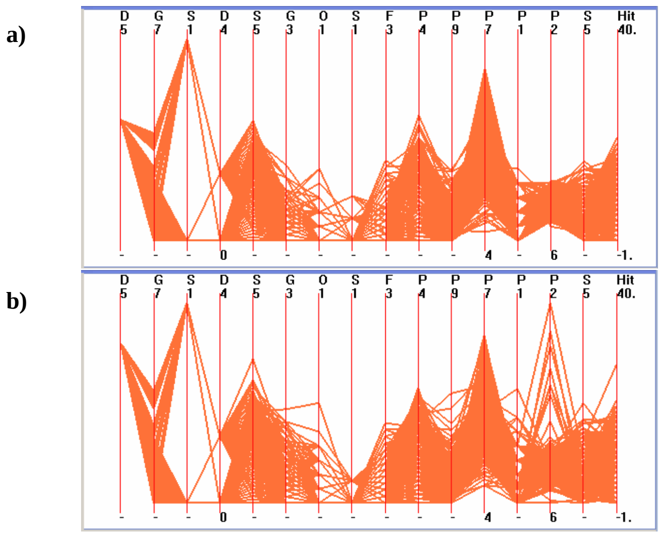


Figure 7: Anomalous spikes found when comparing datasets. The spikes on the right correspond to a real, but elusive bug that occurred. It was eventually fixed of course, but with the ShadyStats, it would've been discovered immediately.

5 Evaluation

I would say that the hierarchical clustering parallel coordinates technique applies very well to the domain of video game statistics analysis. The fact that three issues have been quickly identified with the test datasets highlights the ShadyStats' usefulness. In addition, the tool addresses all of the tasks laid out in section 1.2 at least to some extent, and applies to the scenarios described in the previous section. It was good to see that the trends showed by the ShadyStats correspond to the behaviour intended during development. There was no way to formally prove this without the ShadyStats, and a lot of emphasis was placed on the "feel of the game". Granted, the feel of the game is just as important (if not more) than nicely balanced curves, but the tool adds a concrete viewing window into the overall behaviour, and provides evidence during what were previously "shoulder shrugging" moments: yes, I guess it works.

The key benefits are the great flexibility allowed by the filtering, clustering, dynamic queries, and immediate feedback on all manipulations to the "viewing window". The view canvases are also greatly beneficial for storing certain views that highlight a particular aspect of the data.

There was no time for user evaluation before the deadline of this project, however this will be accomplished in the near future, and work on this tool will continue. Please note that there is nothing video-game specific in this tool. Everything is data driven, and it could easily be applied to any other multi-dimensional data set.

5.1 Strengths

The tool addresses all of the tasks in section 1.2 at least to some degree:

- Shows correlations, trends and relationships through both the high-level parallel coordinates component, and the more detailed scatter plots.
- Helps detect bugs by easily comparing between datasets, as described in 4.3.
- Helps tuning a balanced game, as highlighted by figure 6 and scenario 4.2.
- Shows trends across time, by plotting various measures against the dataset. Please notice in figure 1 that the number of fouls decreases as the dataset ID increases, meaning that the AI was tuned to be more "clean" playing during development.
- The SaveToFile feature of the ZedGraph was a pleasant surprise that can be used to generate pretty pictures for reports, presentations, and what not.

5.2 Weaknesses

The main deficiency of the ShadyStats is its use of coloring. There wasn't enough time to customize it properly, and the default hierarchical cluster coloring of Xmdv does not apply very well to the domain. For example when comparing difficulty trends, each mean line should be a different color, and the aggregate shading should correspond to that color for easy differentiation. Currently, all mean lines are drawn in red, making it difficult to distinguish between them, and the shading color changes with the ordering of the hierarchy. Changing the cluster ordering (ie. moving Side up to the top of the hierarchy for example) changes the cluster colors, even though they represent the same dimension value.

Colors should consistently represent an intended value (for example home, away, different difficulty levels, or difficulty sets) through all clustering orders and detail magnification levels. They should also correspond as much as possible between the parallel coordinates and scatterplot graphs. All of these change almost arbitrarily at the moment. Proper coloring implementation will require a lot of experimentation, and more research done into some formal color theory.

There is also a severe lack of specific values feedback. Moving the mouse over any shaded area or line should give specific information about what it represents: what data set it is from, whether it is home or away, difficulty level, the corresponding dimension value, and the min/max of the aggregate shading area. This may cause problems with overlapping areas, so dynamic highlighting should be applied to the entire shaded region to give immediate feedback of what the information pertains to.

5.3 Performance

Performance is decent, but far from perfect. The downside of implementing the system in the .Net framework is that it instantly eats up about 40 megs of RAM. It would have been a similar story with Java though, so this is not a big deal, since memory is cheap.

The Xmdv parallel coordinates component is the biggest bottleneck at the moment, and it was expected at first that the problem lies with the OpenGL to C# link. However, loading my dataset directly into the unmodified XmdvTool.exe also performed similarly slow while rendering the clusters. It did not seem to be dependent on the number of clusters displayed, because even when displaying one shaded cluster, it behaved almost the same as when displaying 10, and the rendering only sped up when the total dataset size was reduced.

The entire data loaded consists of 4272 rows * 18 columns, equal to 76,896 doubles. It seems that the rendering routine in Xmdv depends on the size of this dataset, when it should only depend on the amount of data actually visible, so there exists an optimization potential here. Converting all the doubles to floats will also result in an instant speed up. We do not need that much accuracy at all, since most of the dimensions correspond to ordinal values.

5.4 Lessons Learned

This has been a very worthwhile project in terms of the material learned, and the practicality of the final solution. It was the first time I have been exposed to alternate visualization techniques such as the parallel coordinate system, which is very useful for this particular domain. It was also the first time I have written Managed C++ interfaces for .Net, and will be able to apply this knowledge in the future.

I have also come to fully appreciate the importance of color, and its interactivity issues when displaying overlaid, transparent data. This is possibly the biggest confusing factor in the clustered hierarchical parallel coordinates component, and will need more investigation and work.

6 FUTURE WORK

A lot of work remains to this first pass of the tool. First and foremost, the "view canvases" need to be fully implemented, and saved to an XML file that can be checked into source control, and easily copied and passed around to other people. The undo/redo system would also be beneficial, and is an easy extension of the view canvases.

A lot more interactivity was planned with the parallel coordinates component, where one could hide, drag-reorder the dimensions, brush and apply filters directly on the dimensions. Xmdv supports some of this interactivity, so more investigation needs to be done into what is already implemented, and what needs to be extended. Better interactivity would eliminate a lot of the clunky list and buttons UI components interfacing to the dimension visibility, ordering and clustering. The first priority however was getting all the necessary features working, and then spending time on improvements, as the tool is used and feedback is gathered.

Being able to select specific records for highlighting in all of the visible graphs was also planned but not implemented due to time constraints. Selecting any data point in any graph, or parallel coordinate control should highlight all of the record's corresponding dimension points in all of the currently visible controls. This will enable easy investigation of outliers that stick out from the norm.

Coloring was identified as a huge weakness of the system, and a considerable amount of time will be spent making it more useful in communicating the data. In addition, the scatterplot layout mechanism is a very naïve, quick and dirty implementation. It would be a useful exercise in investigating the Piccolo.NET [10] library, and if it could handle the requirements of laying out and zooming any arbitrary .Net user control.

The filtering interface can also be greatly improved, as mentioned briefly, earlier. This was again a byproduct of the time crunch, and different interface techniques will be investigated for a more streamlined interface.

7 Conclusion

The field of video game AI is a very challenging and rewarding domain. The goal is much more than creating the uber, unbeatable AI system. It should rather be able to provide challenge for all skill levels, properly communicate the game's puzzles to the player in an interesting manner (through gameplay rather than explicit tutorials), and be able to surprise with different and unpredictable behavior. The best compliment an AI programmer can receive is "it feels human", which means something entirely different than "it is so hard".

AI systems based on probability and randomness achieve a lot of the goals described above, however they can be harder to tune, and their behavior may be more difficult to validate and debug. Overall behavior can only be evaluated and guided through the analysis of statistics. The ShadyStats has been introduced as a highly interactive statistical visualization system with customizable filters, multiple view canvases, hierarchical parallel coordinates for the high-level overviews, and scatterplot graphs created on demand for in-depth analysis of particular dimensions, trends and correlations.

The ShadyStats is meant as a debugging tool first and foremost, and secondly as communication aid between professionals involved in the production of a video game, with a variety of different technical backgrounds and roles: programmers, designers, QA testers, producers and managers. It does not however contain anything video-game specific, and can be used as a general analysis tool on any dataset.

References

[1] Edward J. Wegman. *Hyperdimensional Data Analysis Using Parallel Coordinates*, Journal of the American Statistical Association, Vol. 85, No. 411. (Sep., 1990), pp. 664-675.

[2] Ying-Huey Fua, Matthew O. Ward, and Elke A. Rundensteiner, *Hierarchical Parallel Coordinates for Visualizing Large Multivariate Data Sets*, IEEE Visualization '99.

[3] Jing Yang, Wei Peng, Matthew O. Ward and Elke A. Rundensteiner, Interactive Hierarchical Dimension Ordering, Spacing and Exploration of High Dimensional Datasets, Proc. InfoVis 2003.

[4] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley,1994.

[5] Chris Ahlberg and Ben Shneiderman, *Visual information seeking: Tight coupling of dynamic query filters with starfield displays*, Proc SIGCHI '94, pages 313-317.

[6] M. Sheelagh T. Carpendale, David J. Cowperthwaite, and F. David Fracchia, *Extending Distortion Viewing Techniques from 2D to 3D Data,* IEEE Computer Graphics and Applications, Special Issue on Information Visualization, 17(4), pp 42 - 51, July 1997.

[7] XmdvTool: http://davis.wpi.edu/~xmdv/

[8] ZedGraph: http://zedgraph.sourceforge.net/

[9] ZzzzRangeBar: http://www.codeproject.com/cs/miscctrl/zzzzrangebar.asp

[10] Piccolo: http://www.cs.umd.edu/hcil/jazz/

[11] ILOG Discovery Preview: http://www2.ilog.com/preview/Discovery/