Visual Game Tuning: Integrating Interactive Visualizations into Game Development

A Thesis Presented

by

Markus Elliot Lacay

to

The Graduate School

In Partial Fulfillment of the

Requirements

For the Degree of

Master of Science

In

Computer Science

Stony Brook University

December 2010

Stony Brook University

The Graduate School

Markus Elliot Lacay

We, the thesis committee for the above candidate for the
Master of Science degree, hereby recommend
acceptance of this thesis.

Klaus Mueller – Thesis Advisor, Associate Professor
Computer Science Dept.

Dimitris Samaras, Ph.D. – Chairperson of Defense, Associate Professor,
Computer Science Dept.

Kevin T. McDonnell, Ph.D. – Associate Professor of Computer Science and Chair,
Department of Mathematics and Computer Science, Dowling College

This thesis is accepted by the Graduate School

Lawrence Martin
Dean of the Graduate School

Abstract of the Thesis

Visual Game Tuning: Integrating Interactive Visualizations into Game Development

by

Markus Elliot Lacay

Master of Science

in

Computer Science

Stony Brook University

2010

Game development is a rapidly evolving area of study that blends together a broad spectrum of technical and creative influences. Through their systematic use of rules, mechanics and multimedia assets, games can be thought of as an increasingly complex set of multivariate state data. Often, the interactions between these variable sets and their affect on game behavior are not readily apparent to the system architects at the time of development. These details can easily be overlooked until much later in the development cycle, when the refactoring of their implementations can result in significant overhead. To combat this problem project managers often employ iterative development, early prototyping, playability heuristics, and user studies. Here, an analytic approach is applied towards parameter tuning by visually modeling games as multi-dimensional datasets with respect to other salient game aspects, such as player preference. The motivation behind this research is to explore the use of interactive visual analytics as a tool that can assist game designers in modifying and discovering underlying relationships in their products. It is also hoped that through the use of interactive visualizations, the cost normally associated with quality assurance and play testing in games will be reduced.

Dedicated to my wife Sharon.
Thank you for putting up with my interests.

TABLE OF CONTENTS

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor Klaus Mueller for pushing me towards excellence in visualization, and for supporting my curiosity. I would also like to thank Dimitris Samaras for inspiring others and myself in the pursuit of careers in computer science and game design. Additionally, I would like to extend my gratitude to the entire Computer Science Department at Stony Brook University for providing such a wonderful program that rewards creativity.

The great folks at Applied Visions Inc. helped tremendously by sharing their experiences and providing the type of feedback I couldn’t have gotten anywhere else. I especially want to thank Ken Doris and Frank Zinghini for being supportive of my ideas and helping me grow as a professional. I’ve been inspired greatly by you both.

Most importantly, I want to extend my deepest gratitude to all of my family and friends who have always been there for me. Without you all, I would not have been able to accomplish what I have. Thanks.

<(‘ ‘<) ^(‘ ‘)^ (>’ ‘)>

1. Introduction

The purpose of this thesis is to explore the utility of introducing interactive visualization elements into the game design and testing workflow. Information visualization can be an invaluable asset in helping designers identify relationships amidst high volumes of multivariate data. Using interactive visualizations to modify game state data in real time, both designers and testers can solve problems more rapidly without requiring the involvement of development teams. Ideally, one would use interactive visualizations to guide development and help streamline the performance of the game development life cycle.

Using the development of a conceptual racing game, *Drift!*, to guide the examples presented in Sections 1.2 and 6, we will examine how integrating interactive visualizations into the game design process will assist designers make better decisions. Visual Game Tuning (VGT) - a toolkit that combines several analytic visualizations into an interactive software suite - is an implementation of this concept and will be introduced in the following Sections 3 and 4.

VGT visualizes relationships in game state data and can modify it to suit design needs in real time. A user of VGT can monitor an arbitrary set of game parameters while a player interacts with the game. This allows for a naturally collaborative play testing approach, and eases the transition of the product from development to production. Implementation details for VGT can be found in Section 5.

In Section 2, an overview of the related work in gameplay heuristics and interactive visualization will be presented, in order to provide the background needed for each of the components used in the VGT toolkit. In the following Section 1.1, an overview of the motivation behind this research is given followed by a closer look into the development of *Drift!* and how it's development can be improved upon through the use of Visual Game Tuning.

1.1. Motivation

Typically, there is significant development overhead associated with the testing and revision phases of the software life cycle. However, these phases are in place in order to avoid future cost (Rubey, Browning, Roberts, & SofTech Inc., 1989). Because games are highly specialized types of interactive software, their testing cycles can be more resource hungry than that of traditional software. As a result, there is a high cost associated with next generation game development, and some teams have started to look for ways to reduce this.

Overall, software design methods such as the waterfall model are ill suited for game development due to their inability to cope with changes to the initial design (Bates, 2004, p. 225). Often we see that successful game development is approached in an agile fashion, where the presence of iterative prototyping, revision, and quality assurance testing is critical (Bates, 2004, pp. 218-219). Iterative prototyping suits game design because of its flexibility, and ability to integrate end-user feedback early on in the development cycle.

On titles developed by major publishers, these activities are usually running in parallel with the creation of creative assets such as sound engineering, 3D Modeling, 2D artwork and GUI design. Each of these components contributes significantly to the high cost normally associated with game development. According to a recent set of studies by entertainment analyst group M2 Research, the average development budget for a multiplatform next-gen game is between $18 to $28 million, and development costs for single-platform projects at an average of $10 million (Crossley, 2010).

Though many in the game industry have embraced agile development in response to pragmatic concerns, it can also be thought of as a way to combat rising development costs by producing feedback early on in the development cycle (Kaner, Bach, & Pettichord, 2001). In doing so, game design teams must cope with producing numerous prototype-builds for the quality assurance (QA) staff to test for bugs and playability. The QA staff then works with a software suite to track bugs and make suggestions so that the developers can later investigate unresolved software quality issues (Bates, 2004, p. 177).

Essentially, in order to change anything about a game's internal state, the QA team must depend on the development team's availability to resolve an issue. This can be a source of perpetual frustration for both teams not being able to keep up with the demands of the other. Solutions to this have involved development teams making specialized debugging user interfaces for the QA staff to tune values to their liking. This process is not only inefficient and costly, but error prone and very labor intensive.

In order to streamline the workflow of the tuning and revision process, a new approach must be made that can present multivariate game data and state information in a way that is both useful and accessible to analysts. The hope is that through the use of visual analytic tools, game development teams can avoid situations like these and acquire a deeper understanding of the relationships between seemingly unrelated sets of game state data.

1.2. *Drift!* – A Game Tuning Scenario

Figure 1 – *Drift!* racing game based on provided Unity3D tutorial

Drift! has a number of variables that control everything, from a car's throttle and turning radius to the value of the environment's gravity and friction coefficients. A developer could easily modify several of these variables via their development environment and run the game on their workstation in order to rapidly test the results.

Advanced game engine editors like Unity3D let developers modify game state information in real-time and test the effects on game behaviors. The advancement of debugging tools geared towards developers has made monumental leaps over the past decade, making what was once a very time-consuming process now trivial.

For the sake of this example, let us say that the developer of *Drift!* isn't the one doing the QA testing, as it is normally the case in large-scale commercial software development. The developer must then make a certain set of assumptions regarding what he thinks the QA team would like to see in terms of play behavior, before shipping it off to them for testing. Unfortunately, the set of assumptions that the developer may need to make can often be very specialized. For instance, a developer working on *Drift!* may need to know what the most appropriate turning radius would be for simulating the feel of a 1992 Mazda RX-7. At this point, the developer can make his best guess and wait for response from the QA staff. The QA staff can then work with a subject matter expert that will ultimately give them some notion of how "off" their initial results are. QA then logs this information in the bug-tracking database for the developer to address when he can allocate enough time away from his other tasks. This cycle will repeat until the correct value is found with the cooperation of the developer and QA staff. Throughout this process, each step depends on the last, and each iteration builds upon the knowledge of the last. In this way, it is similar to the familiar waterfall software design paradigm that was discussed in Section 1.1 and shown to be quite contrary to successful game development.

One potential solution to the inefficiency in this process would be for the developer to create a user interface for the QA staff so that they can tune the values in question to their liking, as was briefly mentioned in Section 1.1. However, this requires that the developer anticipate which values may be requested for modification, requiring additional QA resources in order to avoid making superfluous debugging interfaces. To avoid this, the developer could simply wait for the QA team to request a specific debugging UI, but this approach also suffers from being time-consuming and labor intensive. Currently, there is no industry standard in approaching such situations, and each development team handles how to deal with this individually.

2. Related Work

Games and their effects on players have been studied at length by a number of individuals that wish to understand the deeper meaning behind player preference, and how this can guide design decisions. As will be discussed in Section 2.1.1, various approaches exist for quantifying the player experience in a way that is suitable for analysis. Starting from this perspective, it is clear that when these models become mature enough, visualization of their data will be critical in using them to guide design decisions.

The adoption of multivariate information visualization into the software design process has been on a steady rise with the proliferation of readily available visualization libraries such as Protovis (Section 5.2.1). Many IDE packages now come with a performance-tuning tool that affords users the ability to visually inspect software performance via scatterplots, bar graphs and other statistical visualizations. Using this same approach, there are now game suites that package informative performance evaluation tools in their editors, as is done in Unity3D game engine shown in Figure 2 and described in Section 5.2.2.

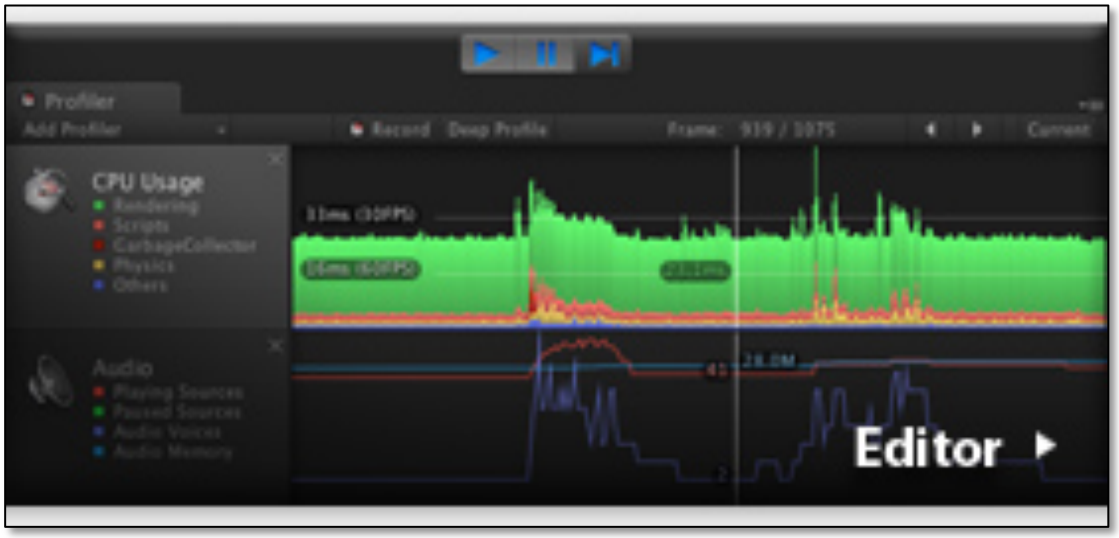


Figure 2 - Unity3D integrated development environment shown displaying performance graphs

In the following sections, several projects that have greatly inspired the creation of Visual Game Tuning are presented. We begin with an approach to playability heuristics followed by a survey of several relevant visualizations.

2.1. Playability Heuristics

"User testing is the benchmark of any playability evaluation, since a designer can never completely predict user behavior" (Desurvire, Caplan, & Toth, 2004). In the software productivity industry, heuristics have served as a way to evaluate the usability of user interfaces with the goal of making them easy to learn, use, and master. In contrast, the goals of game design can be characterized as "easy to learn, difficult to master (Malone, 1982)." Drawing the line between what is intuitive to players versus what is complementary to gameplay is a challenging task that requires knowledge of the systems involved. As a result, game designers are faced with difficult design decisions that can only be made with the assistance of play testers. However, the effectiveness of using play testers to guide design is limited by their ability to express their preferences in a way that is conducive to refining play models. In the following Section 2.1.1, an approach to this issue is addressed.

2.1.1. Heuristic Evaluation for Playability (HEP)

The HEP method was born out of the need for a standardized way to evaluate game player feedback by focusing on the relevant concerns of gaming software. It was developed with the intention of providing an analytic criterion by which designers can evaluate potential pitfalls early on in the development cycle. Prior to this study, Malone (Malone, 1982), and later Federoff (Federoff, 2003) (Federoff, 2002), compiled a list of game heuristics for general and educational video games. HEP evaluates playability by defining four distinct game heuristic categories as the following: *"game play* is the set of problems and challenges a user must face to win a game; *game story* includes all plot and character development; *game mechanics* involve the programming that provides the structure by which units interact with the environment; and *game usability* addresses the interface and encompasses the elements the user utilizes to interact with the game (e.g. mouse, key-board, controller, game shell, heads-up display)" (Desurvire, Caplan, & Toth, 2004, p. 1509) .

	Heuristic and Description
Game Play - 15	Pace the game to apply pressure but not frustrate the player. Vary the difficulty level so that the player has greater challenge as they develop mastery. Easy to learn, hard to master.
Game Story - 5	The Player has a sense of control over their character and is able to use tactics and strategies.
Mechanics - 1	Game should react in a consistent, challenging, and exciting way to the player's actions (e.g., appropriate music with the action).
Usability - 1	Provide immediate feedback for user actions.

Table 1 - Sample set of Heuristics for Evaluating Playability (HEP)

It is through these game heuristic categories that one can infer correlations between user preferences and changes to the game. However, this evaluation alone cannot specifically determine the changes to the design needed in order to fulfill the desired playability result. For instance, the first usability heuristic can be interpreted in a number of ways that depend on the player's preference, designer's intentions, and other more complex performance factors. Desurvire's study found that overall, HEP proved to be a valuable asset in benchmarking the playability of games. While uncovering many of the same results, the user study's findings highlighted specific issues in the design - forming a complementary pairing of both analytic and empirical methods (Desurvire, Caplan, & Toth, 2004, p. 1512).

HEP provides a valuable example showing how quantitative user preference analysis can help guide game design. In the example of designing *Drift!,* HEP could be used to quantify player feedback in a way that can easily integrate into VGT's indicator constructs – described later in Section 4.1.5.

2.2. Interactive Visualizations

Edward Tufte is quoted as having said, "[When] you see excellent graphics, find out how they were done. Borrow strength from demonstrated excellence. The idea for information design is: Don't get it original, get it right" (Tufte, 2001). This is the spirit in which this thesis approaches the subject of visualization. The visual techniques described in the following sections were picked precisely because of their concise expressiveness and applicability to the problem domain described by Section 1.1. The motivation behind Visual Game Tuning was greatly inspired by the following three studies, as they collectively represent innovation in quality interactive visual analytics.

2.2.1. Interactive Dimension Reduction through User-Defined Combinations of Quality Metrics (IDR)

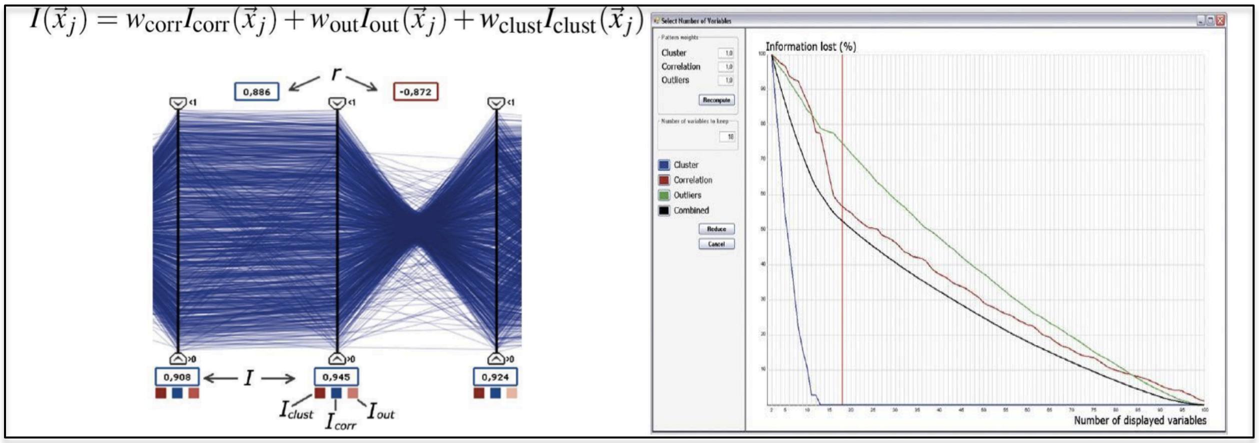


Figure 3 - Interactive Dimension Reduction uses weighted metrics (left) to determine the information loss (right) for evaluating the importance of variables and the relative impact of removing them from the visualization.

IDR uses a combination of parallel coordinate graphs and numeric metric representation in order to assign importance values to multivariate sets of data. The motivation behind the study was to find which variables best represented a dataset so that an analyst can reduce the volume of visual information needed in order to accurately represent large multivariate datasets (Johansson & Johansson, 2009). The value in this approach is that an analyst can more easily compare and contrast the effective "importance" of a single dimension in relationship to the rest of the data set quantitatively by minimizing the "information loss".

In the study, three metrics - Pearson Coefficient, Outliers, and Clustering - were chosen to gauge the importance of a variable. The weighted sum of these values was then computed and presented to the user in a seamless manner by embedding the metric and importance values into the parallel coordinate graph. The user is then allowed to interactively modify which data is visually represented by using the information loss graph as a guide towards reducing visual clutter, as is depicted in **Figure 3.** A related approach that uses correlation metrics to assign variable importance will be used in VGT's detail panel between data sets.

2.2.2. *Scattering Points in Parallel Coordinates (SPC)*

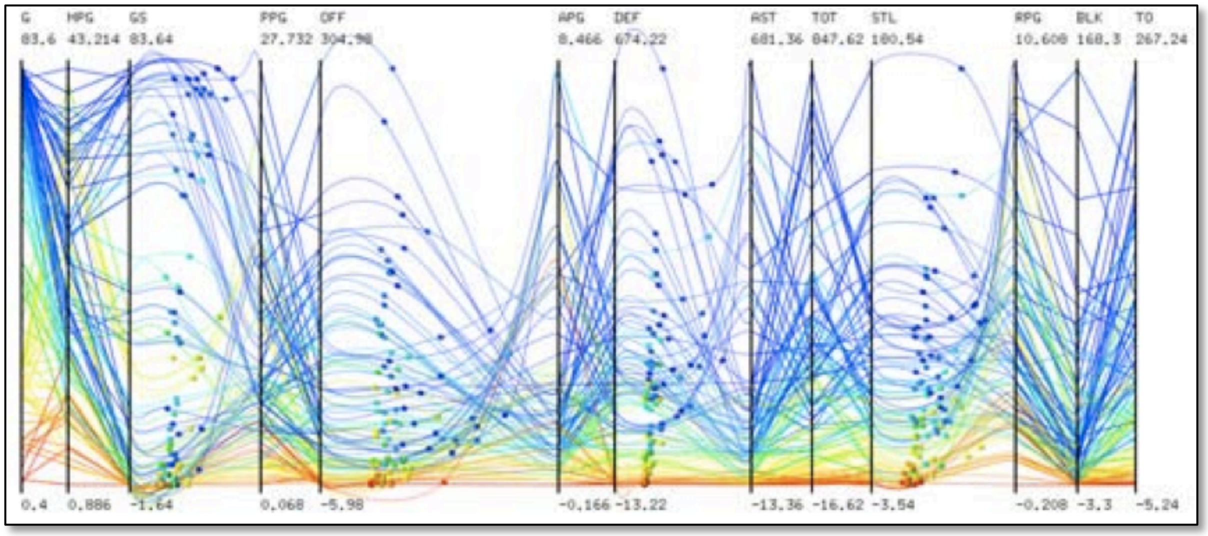
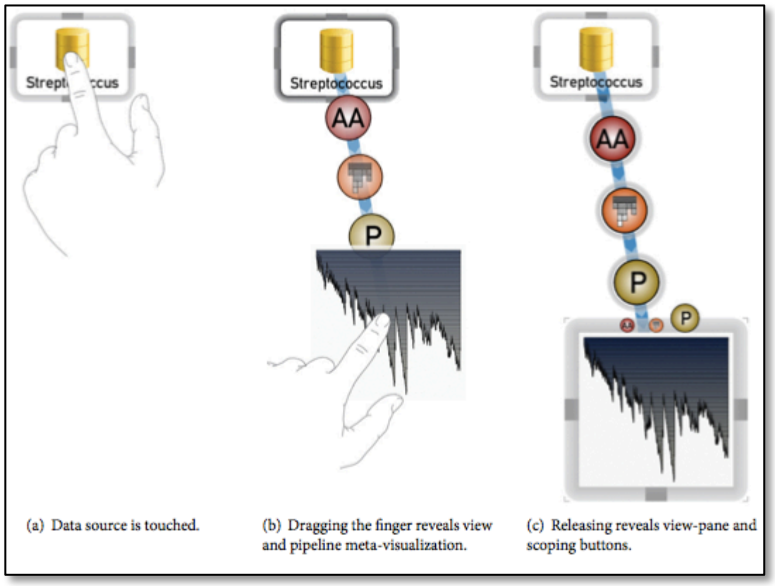


Figure 4 - SPC depicted showing the concise manner in which it can express clustering and variable relationships

When visualizing multivariate data, the dilemma of deciding which data relationships to emphasize is of paramount importance. For instance, Parallel Coordinate graphs effectively establish the correlative relationships between a set of dimensions, as they were designed with this purpose in mind (Andrienko, 2001). However when analyzing Parallel Coordinate graphs, relationships can be difficult to detect for numerous densely populated datasets – requiring some metric or numeric analysis to guide inferences. Though when trying to understand numeric analysis, a scatterplot may be necessary in order to ensure the integrity of the metric is not being skewed by outlier or clustering effects - as is the case when using the Pearson Coefficient as a metric (Wilcox, 2005).

Yuan et al. provide an effective visual solution to the dilemmas described above by integrating scatterplots between the dimensions of a parallel coordinate graphs. In their approach, an effective means of integrating the two through novel use of Bezier curves on the parallel coordinates is described (Yuan, Guo, Xiao, Zhou, & Qu, 2009). This approach has the distinct advantage of being visually compact, relatively easy to analyze, and expressive of metric accuracy. VGT uses a similar approach in its detail panel in order to promote more informed analysis.

2.2.3. *Lark: Using Meta-Visualizations for Coordinating Collaboration*



Streptococcus
Streptococcus
AA
P
Streptococcus
AA
P
P
(a) Data source is touched.
(b) Dragging the finger reveals view and pipeline meta-visualization.
(c) Releasing reveals view-pane and scoping buttons.

Figure 5 – Lark's interaction technique for creating new visualizations by interacting with existing ones

The allure of collaborative visualization concepts is gaining popularity due to its vision of allowing multiple analysts to work in concert on a dataset. However, most visualization is not developed with parallel interaction in mind, and so there exists a need for designing new interfaces that can accommodate collaborative visual analysis. Tobiasz's exemplary work on Lark approaches this problem in a novel way that integrates multiple visualization techniques with advancements in Human Computer Interface (HCI) via a tabletop interface (Tobiasz, 2010).

The motivation behind creating Lark was to assist small groups engaging in synchronous co-located collaborative data analysis. By observing non-digital collaborative processes, Tobiasz highlights what he found to be "three important concepts often found in mixed-focus collaborative work: scoped interaction, temporal flexibility, and spatial flexibility" (Tobiasz, 2010, p. 7). He describes it as "the unconstrained spatial organization and mobility of artifacts in and people in and around the work environment, allowing each analyst to organize him/herself in the workspace however he/she feels is most appropriate", *scoped interaction* as "the extent of an action's effects as controlled by the performer of the action"; and *temporal flexibility* as "the unrestricted temporal ordering of activities." (Tobiasz, 2010, pp. 8-10)

Lark was built using a large digital tabletop with these concepts in mind, and took advantage of interactive visualizations because of their intuitive ability to elaborate information in a "details on demand" fashion (Shneiderman, 1996). It was found that because of Lark's natural interaction and collaborative abilities, it showed potential in helping a group process become more efficient. In effect, Lark's success played a large role in inspiring the creation of Visual Game Tuning.

2.3. Summary

The discovery of gameplay heuristics has made it possible to begin quantifying the effects that user preferences have on game design. It is through this lens of analysis that we begin to see the utility of visualizing these quantities in ways that are beneficial to guiding designs. Advancements in interactive visualizations have made it possible for individuals to have full command over large sets of high-dimensional data with relatively little expertise. By starting out with the idea that interactive visualizations can be used to express and control gameplay, the motivation behind Visual Game Tuning has been established. The following section begins to explore the viability of integrating interactive visualizations into the game design process.

3. VGT: Visual Game Tuning

VGT is a cross-platform visual toolkit designed to help game developers interactively discover and modify relationships between variables that dictate game behaviors. It can be thought of as a “virtual mix board” for game designers who wish to remotely tune a game’s parameters in real-time without the need for a development environment. The following Section 3.1 begins with an overview of the VGT toolkit, and is followed by a description of how it fits in to the overall game design process.

3.1. Overview of VGT

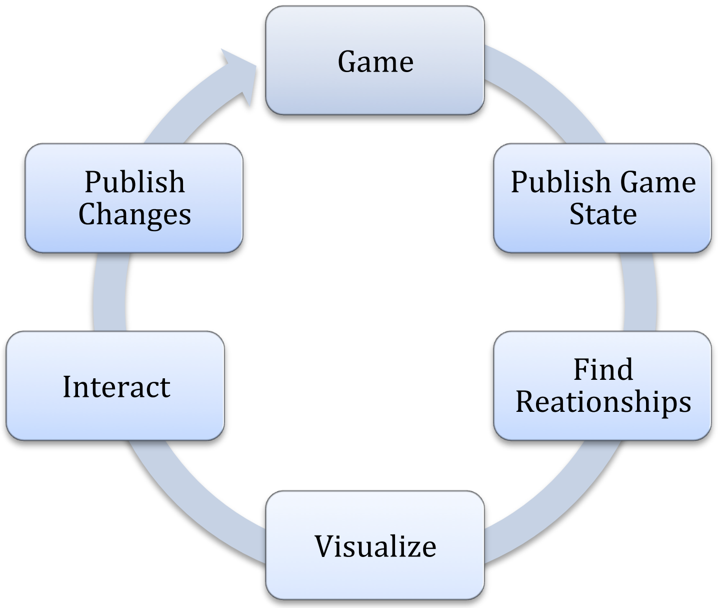


Figure 6 - Overview of the VGT runtime cycle

The VGT toolkit consists of several components that deal with exposing, modifying, analyzing and visualizing game parameters. VGT is intended for use between two parties: the first being that of the player who will be driving the games, and the other being that of the VGT user who will be analyzing and modifying the game environment. Shown in Figure 6 is the flow of VGT’s runtime, which starts with a game publishing its parameter information to a graphical client. It is here that relationships between variables are analyzed and presented to the VGT user in the form of multiple visualization and control panels. Once the VGT user has completed analysis of the game’s current parameters, he can then apply changes to them directly. At this point, the changes are then published back to the game, where it will update its parameters. Further details on the specific implementation of this interaction can be found in Section 4. Following here is a walkthrough of the game design process in relation to VGT.

3.1.1. Test guided tuning

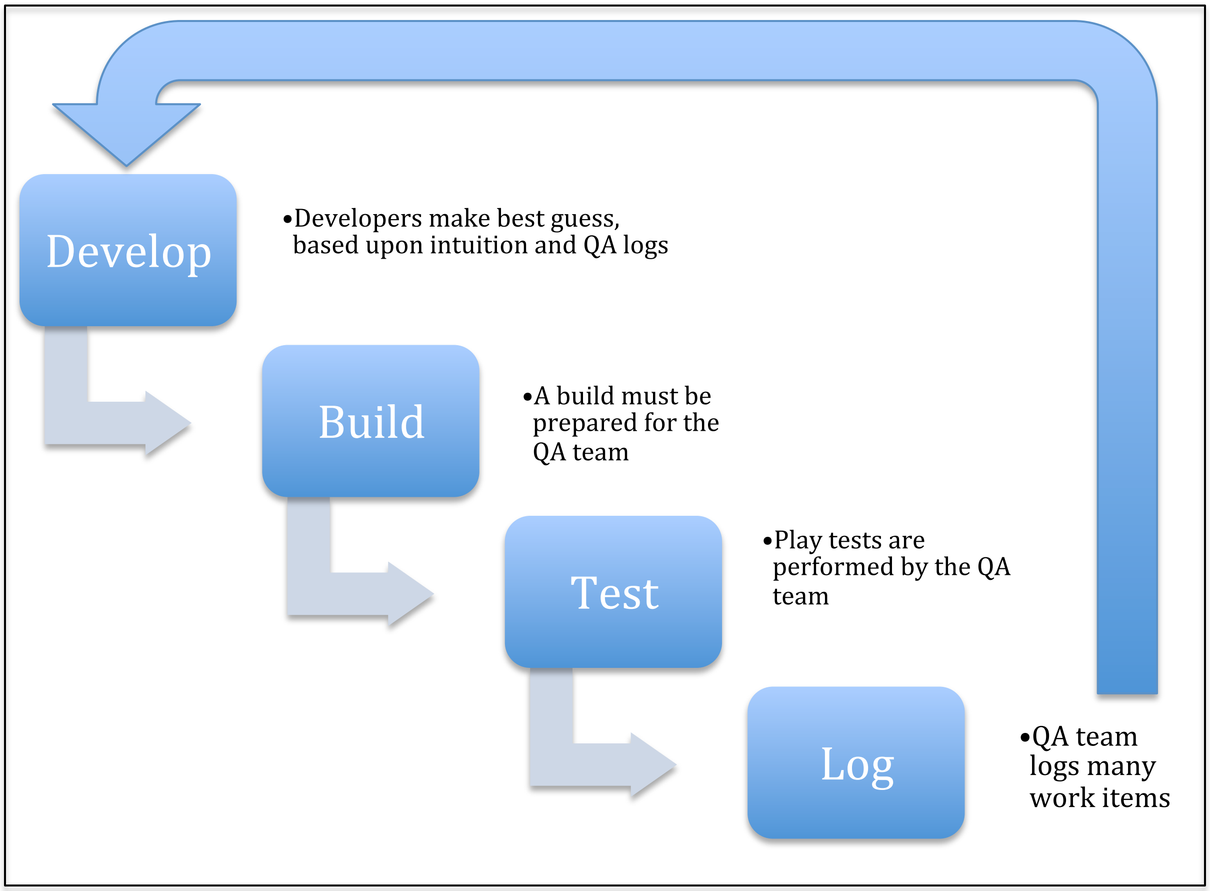

Develop
•Developers make best guess, based upon intuition and QA logs
Build
•A build must be prepared for the QA team
Test
•Play tests are performed by the QA team
Log
•QA team logs many work items

Figure 7 - Typical waterfall QA game test cycle

Test guided tuning, or quality assurance testing, performs the critical function of ensuring that games are developed in a manner that reduces gameplay inconsistencies and software bugs. In smaller development teams this process is usually carried out by the developers themselves, however in high-profile commercial games, full-time QA departments are used. (Bethke, 2003, p. 52) Often, the testing process can begin as soon as the first implementations of the game are produced (Bates, 2004, p. 176). At this early stage, the QA team can begin monitoring progress until as late as post-production (Bethke, 2003, p. 53). As mentioned earlier, QA staffs often employ bug-tracking systems that allow the development team to easily access work items as they are reported. Throughout this process, a considerable amount of collaborative effort is needed in order to properly interpret the testing results. Often, there will be an assigned lead tester who meets with the development and production leads on a daily basis to discuss these issues (Bates, 2004, p. 179).

Visual Game Tuning can act as a basis for a more immediate feedback loop in the game testing and revision cycle. For example, in the testing process depicted in Figure 7, there is a layer of separation between the QA staff and the development team. The QA staff can find a bug, document the circumstances that reproduce it, and submit it for review. The time spent on this process is usually minimized by use of efficient bug tracking systems, however it requires the interpretation of developers and their respective project leads. If one were to assume that game developers have tasks other than fixing software bugs, as is often the case, patches can be pushed off until later in the development cycle where they tend to cost more to implement (Rubey, Browning, Roberts, & SofTech Inc., 1989).

Using VGT, QA and design leads can take matters into their own hands by providing an informative interface into the game state that they can analyze, modify and log for later fixes. As shown in the following section, VGT allows individual developers to spend less time interpreting testing results and reduces the ambiguity in which QA teams can express the desired changes. Interactive information visualizations are the ideal format to allow game architects and QA staffs to abstractly interact with multivariate game state data in a way that is both informative and natural.

3.1.2. Visually guided analysis

Capitalizing on recent advancements in information visualization, the capability to intuitively depict large multivariate data is greater than ever. Tuning the parameters of a real-time simulation that emulates the behavior of realistic world physics, gameplay, and narrative elements can be a very complex task due to its multivariate nature. Despite this, game developers are often tasked with understanding exactly how each of these variables interacts with one another. QA testers also need to be aware of the underlying architecture in order to provide detailed commentary for developers to assimilate into future builds. In game testing, this information is often passed along to the development staff in the form of QA logs and formal reports (Bates, 2004, p. 178).

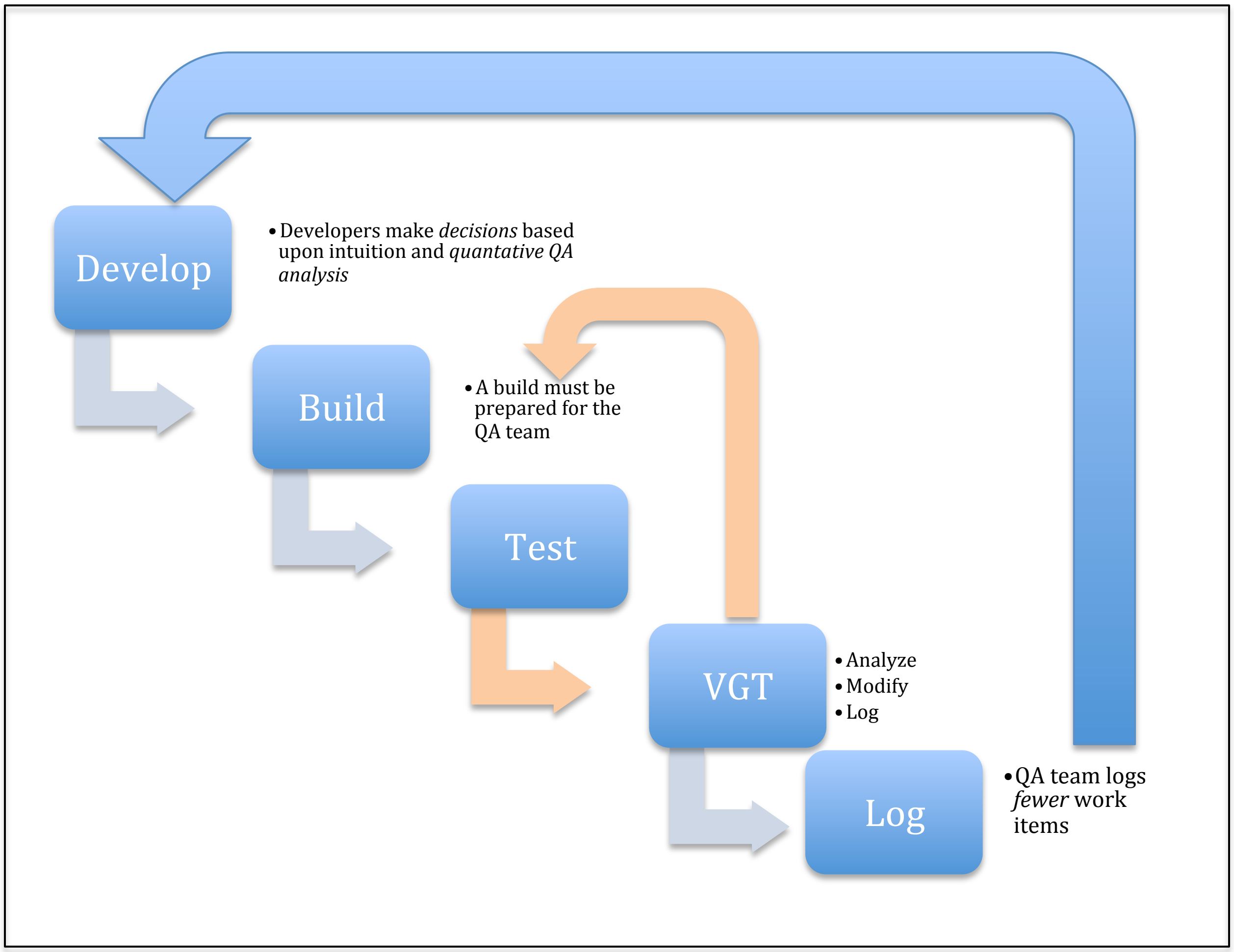

Develop
• Developers make *decisions* based upon intuition and *quantative QA analysis*
Build
• A build must be prepared for the QA team
Test
VGT
• Analyze
• Modify
• Log
Log
• QA team logs *fewer* work items

Figure 8 - VGT modified QA game test cycle

Typically, information about bugs, player preferences and comments are processed in a waterfall manner as was shown in the previous section. Providing insight into produced work items is left up to the QA staff's ability to document them in a way that the development team can use to make design decisions. Developers then use these incident reports as a guide for what to focus on when refactoring. In many instances, this entire interaction could have been avoided if the game tester had an intuitive means of accessing game-state information to analyze and modify. This is where interactive visualizations become key.

Using an interactive visualization a game tester can select multiple parameters to investigate, compare, and detail what the state of each selected parameter was at a particular point in time. In the state detail, the visualization can expose numeric analysis metrics in order to let the tester know that there is a potential relationship between two parameters should he wish to modify one of them. This metric can be setup to perform tracking and logging for later analysis if observed to be within a certain threshold. Armed with this knowledge, a tester can now modify the game-state with some additional insight and observe results that he can report to the development team. This approach has the distinct advantage of using interactive visualizations as a way to reduce the communication burden between QA and development staffs. It is also the goal that through the use of customized metrics, game developers will become more aware of the systemic effects independent variable changes will have on the simulated environment as a whole.

3.1.3. Collaborative tuning

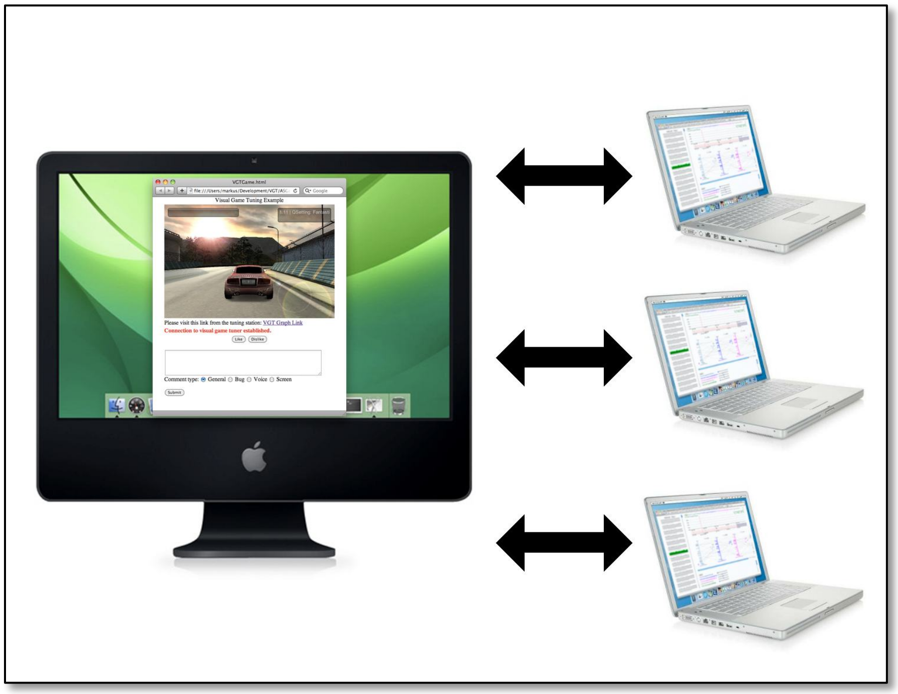


Figure 9 - Conceptual VGT collaboration shown using multiple *vis* clients

Tobiasz's work on his collaborative visualization framework, Lark, showed the value in retrofitting group processes with visually collaborative techniques (Tobiasz, 2010, p. 123). More specifically, Visual Game Tuning approaches the topic of collaboration with the vision of multiple game designers being present during playtests with the ability to modify the same parameters. Using VGT, more than a single individual can take part, allowing QA, development, and design staffs to cooperate in unison during playtests. This alleviates the burden that QA and development staffs usually encounter when trying to understand each other's workflow.

3.2. Summary

In this section, it has been shown how Visual Game Tuning fits into the overall game development cycle. It has also been shown that the traditional test guided tuning model has inherent inefficiencies and may potentially benefit from collaborative visually guided models. It can be seen that interactive visualizations have the potential to act as intuitive interfaces into complex game-state data, making it easier to analyze. In the following Section 4, the specific interactions of the VGT toolkit are detailed within the context of the topics discussed.

4. VGT Interaction

VGT requires that there be a running game providing parameter data to the graphical frontend. Naturally, this splits a VGT session into two clients: *vis* and *game*.

The VGT *game* client can be as simple as a game environment configured to publish and receive its parameter information over a network connection. Though developers can extend this to additional interfaces that can be used to submit preferences, bugs, and comments as they occur.

The VGT *vis* client follows Shneiderman's *Visual Information-Seeking Mantra,* "Overview first, zoom and filter, then details-on-demand" (Shneiderman, 1996). It does this by providing users with a graphical interface that intuitively displays a set of parameters over time where they can be selected for more detailed analysis and modification. The *vis* interface can be broken down into seven parts: *Overview, Zoom & Filter, Detail, Relationship, History, Extract,* and *Update*.

Following this, the VGT-*vis* client's interface is presented and broken down by component. Afterwards, a sample game client is presented exploring the possibilities for additional interface options.

4.1. VGT-*vis* client

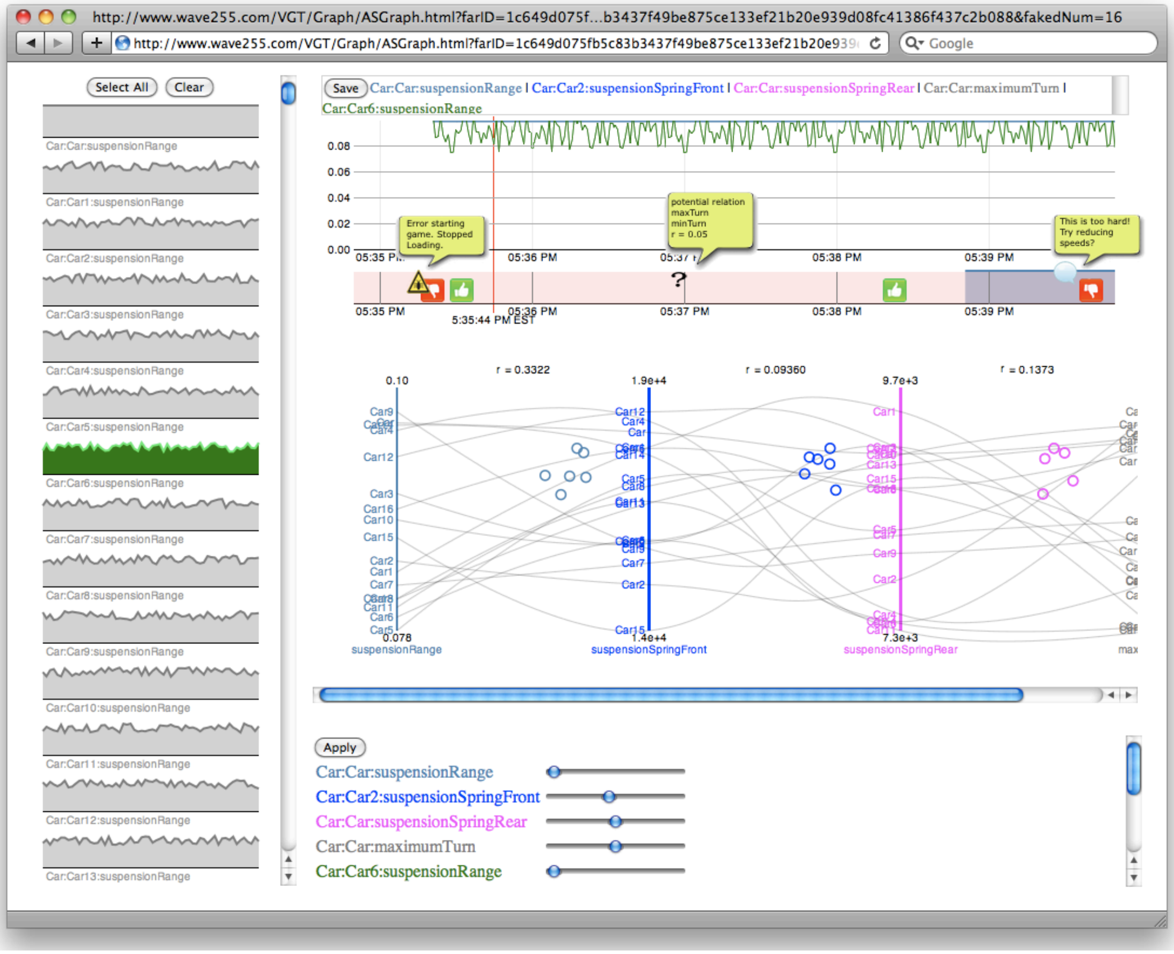

http://www.wave255.com/VGT/Graph/ASGraph.html?farID=1c649d075f...b3437f49be875ce133ef21b20e939d08fc41386f437c2b088&fakedNum=16
Select All
Clear
Save
Car:Car:suspensionRange | Car:Car2:suspensionSpringFront | Car:Car:suspensionSpringRear | Car:Car:maximumTurn |
Car:Car6:suspensionRange
Error starting game. Stopped Loading.
potential relation maxTurn minTurn r = 0.05
This is too hard! Try reducing speeds?
5:35:44 PM EST
r = 0.3322
r = 0.09360
r = 0.1373
suspensionRange
suspensionSpringFront
suspensionSpringRear
Apply
Car:Car:suspensionRange
Car:Car2:suspensionSpringFront
Car:Car:suspensionSpringRear
Car:Car:maximumTurn
Car:Car6:suspensionRange

Figure 10 - Overview of VGT's graphical interface

4.1.1. Overview – Visual Parameter Histories

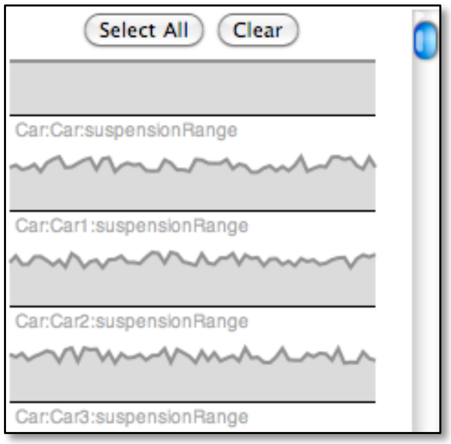


Figure 11 – *vis* client Overview panel showing a scrollable list of in-game parameter histories with colors denoting those the user selected for analysis.

This list serves as the starting point for analysis by allowing the user to have some insight into the available datasets, and select which he is specifically interested in analyzing. Card, Mackinlay and Shneiderman's emphasis on starting with an overview as a "general heuristic of visualization" is underscored in *Readings in Information Visualization* where they comment on its ability to reduce search time, improve detection of overall patterns, and assist the user in choosing their next move. (Card, Mackinlay, & Shneiderman, 1999)

4.1.2. Zoom & Filter – Using combined Focus+Context (F+C) Line Charts

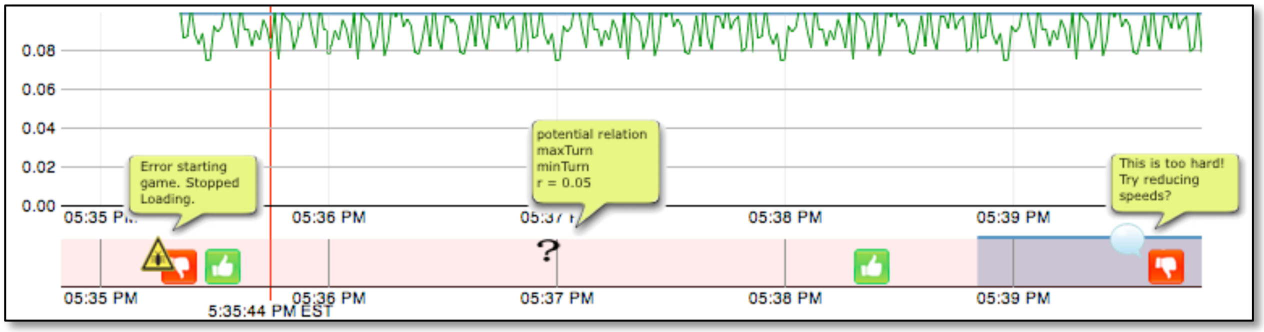

0.08
0.06
0.04
0.02
0.00
05:35 PM
05:36 PM
05:37 PM
05:38 PM
05:39 PM
Error starting game. Stopped Loading.
potential relation maxTurn minTurn r = 0.05
This is too hard! Try reducing speeds?
?
05:35 PM
05:36 PM
05:37 PM
05:38 PM
05:39 PM
5:35:44 PM EST

Figure 12 – *vis* client Zoom & Focus panels shown using a Focus + Context approach using multiple line graphs to represent datasets

Though the most well known F+C approaches use Distortion Techniques (Hinum, 2006, p. 17) the same functionality can be provided in a dual layer line chart where one chart acts as an overview of the data and the other as the zoomed data in focus. In the Zoom

& Focus panels shown in Figure 12, the bottom Zoom panel acts as the overview of the data and has a “draggable” region that the user can modify in order to control the Focus panel above it. The focus region takes up a larger potion of the view space in order to let the user investigate details more accurately. Here, the lines are colored according to the selection color made in the Overview panel, and labeled above the F+C charts in the Group panel. For each parameter selected in the Overview panel, one colored line will represent its value with respect to time. Embedded in the F+C chart, we have an index line that follows the user’s cursor while hovering. This index line sets up the following Detail panel.

4.1.3. Detail – Using Time-Oriented Information

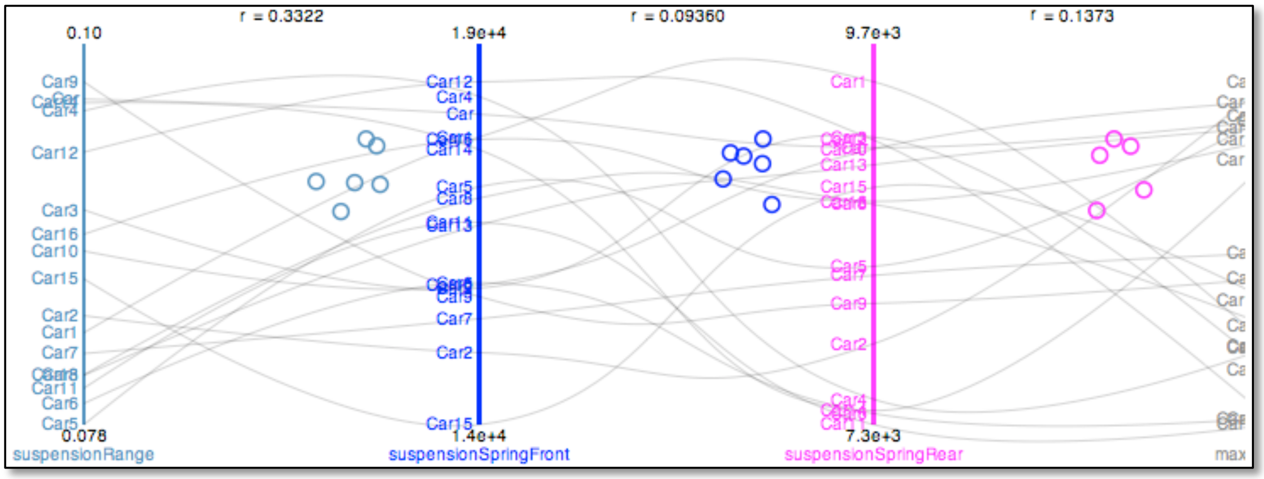


Figure 13 – *vis* client Detail panel shown using temporally indexed parallel coordinate graphs visualizing scatterplots and metrics between dimensions

Using a Time-Oriented Information approach (Hinum, 2006, p. 17) the detail panel uses the temporally indexed focus line, detailed in the F+C panel above, to focus on the state of selected parameters and their relationships at a given point in time. As shown in Figure 13, a parallel coordinate graph is shown to the user, displaying what the values of each parameter dimension were for the time interval selected. This panel is special in that it will automatically aggregate all other game objects sharing the same field into displayed values along each parallel dimension. This visualization was inspired by the work of Yuan et al. where scatterplots are introduced between dimensions in order to give insight into the relationship between two adjacent dimensions (Yuan, Guo, Xiao, Zhou, & Qu, 2009). The

Detail panel defaults to using the left dimension as the vertical axis and the right dimension as the horizontal axis for each dimension pair. The details of the numerical analysis that this panel compliments are discussed in the next section.

4.1.4. Relationship – Using Numerical And Visual Analysis

As mentioned in the description of the Detail panel, scatterplots are useful to find relationships and trending between parameters, however, can also be useful to use metrics such as the Pearson Coefficient to find linear correlation between potentially similar datasets (Johansson & Johansson, 2009). This metric becomes especially useful when inspecting densely populated datasets, such as most multiplayer game states, where relationships may not be readily apparent by visually looking for trends. To accomplish this, each dataset automatically computes its own correlation coefficient and displays this value, denoted by *r*, between each pair of dimensions. However, *r* only becomes a useful metric within a certain threshold that is specific to the problem domain. If the correlation coefficient, or any other metric, is found to be within a user-defined threshold, alerts can be presented to the user, as is the case in the following section on using indicators.

4.1.5. History – Using Indicators

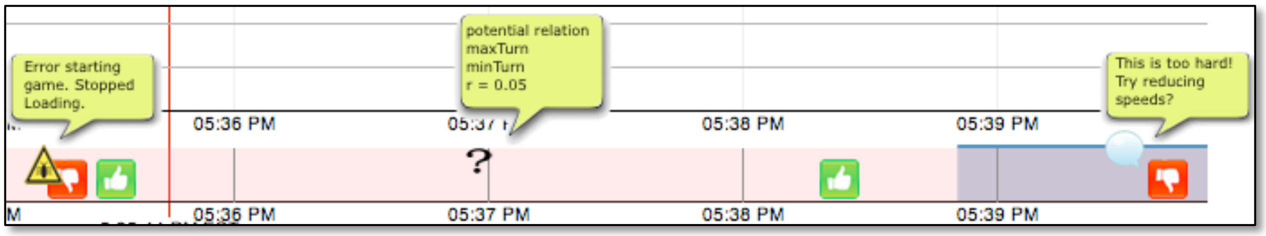

Error starting game. Stopped Loading.
potential relation maxTurn minTurn r = 0.05
This is too hard! Try reducing speeds?
05:36 PM
05:37 PM
05:38 PM
05:39 PM

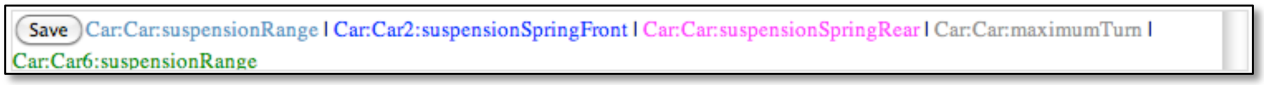
Figure 14 - vis client Focus panel showing submitted player preference, bug, and comment reports

In the Detail panel metrics - such as the Pearson Coefficient - are computed for each selected time interval. Using a user-defined threshold, the Detail panel alerts the Zoom panel by using an indicator that will be logged and associated with a timestamp. These indicators will provide information to the user via icons and cursor-activated tooltips.

As shown in Figure 14, the user is made aware that there is a potential relationship between two selected parameters at a particular point in the game's runtime. This time is now permanently catalogued and associated with the relationships between the relevant parameters.

Indicators need not be restricted to assisting relationship analysis. They can represent user preferences that are discovered via player input on the *game* client. Players can be provided with integrated preference and bug tracking interfaces that make it simple to share their experiences in the game as they occur. This data can then be compiled and analyzed alongside other changing parameter information, in order to draw inferences between player preference and game state variables. The sample interface to acquire this data can be found in Section 4.2 where the *game* client's interaction is detailed.

4.1.6. Extract – Using persistent groupings of data



Car:Car:suspensionRange | Car:Car2:suspensionSpringFront | Car:Car:suspensionSpringRear | Car:Car:maximumTurn | Car:Car6:suspensionRange

Figure 15 - vis client Group panel showing the group of active parameters

Using gathered insight into the relationships between variables, one can save findings for future comparison. To do this, a Save is provided to name a set of parameters as a focus group. Later on, this focus group can be used to compare with other parameters and extract new relationship inferences. By analyzing these inferences, one can more accurately gauge the systematic effect of changing any of the selected parameters. Using this information, one can then commit desired changes to the game state by using the Commit panel.

4.1.7. Update – Using standard GUI controls

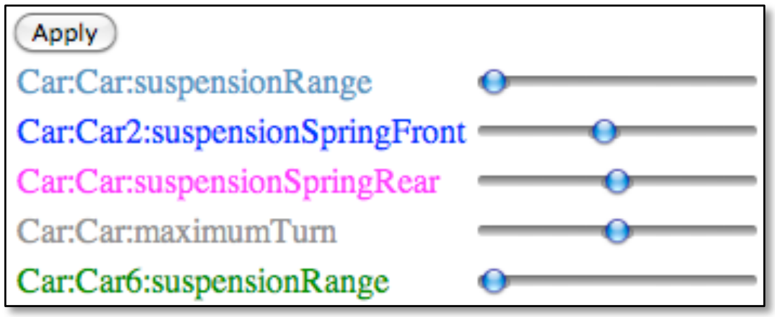
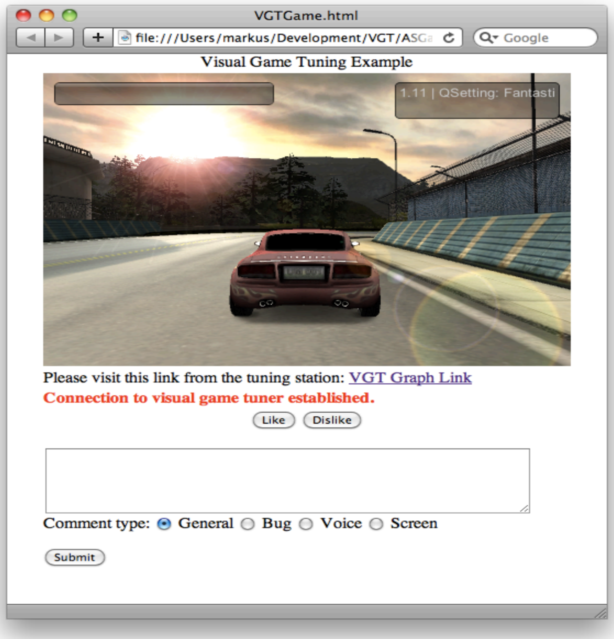


Figure 16 – *vis* client Update panel shown allowing users to modify game parameters through slider controls

VGT's primary functions are to allow game testers to analyze and refine their models in real time. Because of this focus on real time interaction, the interface to modify game parameters is a simple set of valued sliders, as many people with minimal computer literacy feel comfortable using them as a way to manipulate data. This approach is in close relationship with audio software packages that model their interfaces to mimic physical mix boards. In this manner, VGT users can very easily manipulate game state values, as shown in Figure 16. Once the user has adjusted parameter values to his liking, he then presses the apply button and the *vis* client will publish changes to the *game* client.

4.2. VGT-*game* client


VGTGame.html
file:///Users/markus/Development/VGT/ASG
Google
Like
Dislike
Comment type: General Bug Voice Screen
Submit

Visual Game Tuning Example

1.11 | QSetting: Fantasti

Please visit this link from the tuning station: VGT Graph Link

Connection to visual game tuner established.

Comment type: [✓ General] Bug Voice Screen

Figure 17 - VGT-*game* client shown

The VGT-*game* client provides a thin layer of optional interaction between the game's players and its designers. In the implementation example shown in Figure 17, the player is presented with the game paired with a very simplistic preference, comment, and bug-reporting interface. This GUI will allow the user to submit their preferences, in the form of a like or dislike, button in addition to bugs and comments as they happen in the game. Should the developers want to support faster reporting, hotkeys can be assigned to game actions that will perform these functions. The motivation behind providing players with preference controls is to facilitate the creation of preference statistics that can be assimilated in real-time side by side with usage information. Through dynamic analysis of user preferences, developers can make more informative decisions about a game's design on the fly. The *game* client's web-interface is optional and only provides an interface to scripting elements. As will be seen in the following Section 5, the implementation of VGT relies on a variety of technologies working together, but does so in a way that is non-intrusive to the game development pipeline.

5. Implementation of VGT

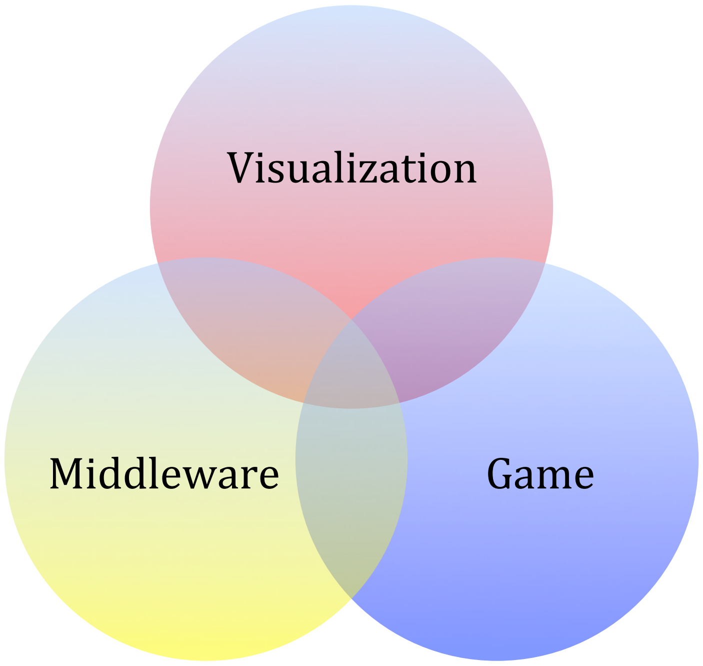


Figure 18 - The three primary components of VGT

VGT consists of three primary components that stand independently of a game in order to monitor it. This interaction depends on the cooperation of a number of technologies. To keep the implementation simple, a web-based approach using the Protovis (Section 5.2.1) visualization library was used for the *vis* client (depicted in Figure 18 as red) while the Unity3D game engine (Section 5.2.2) was used for the *game* client (depicted in Figure 18 as blue). For communication between these two components, the middleware used (depicted in Figure 18 as yellow) was a rudimentary peer-to-peer connection using RTMP and ActionScript (Section 5.2.3). Ideally, one would implement this component as a managed TCP connection to a database server designed for this purpose but this was unnecessary as a proof of concept.

The stages involved in each of the three components are presented in Figure 19. Following this figure, the specifics of how each stage is implemented are detailed in the following sections.

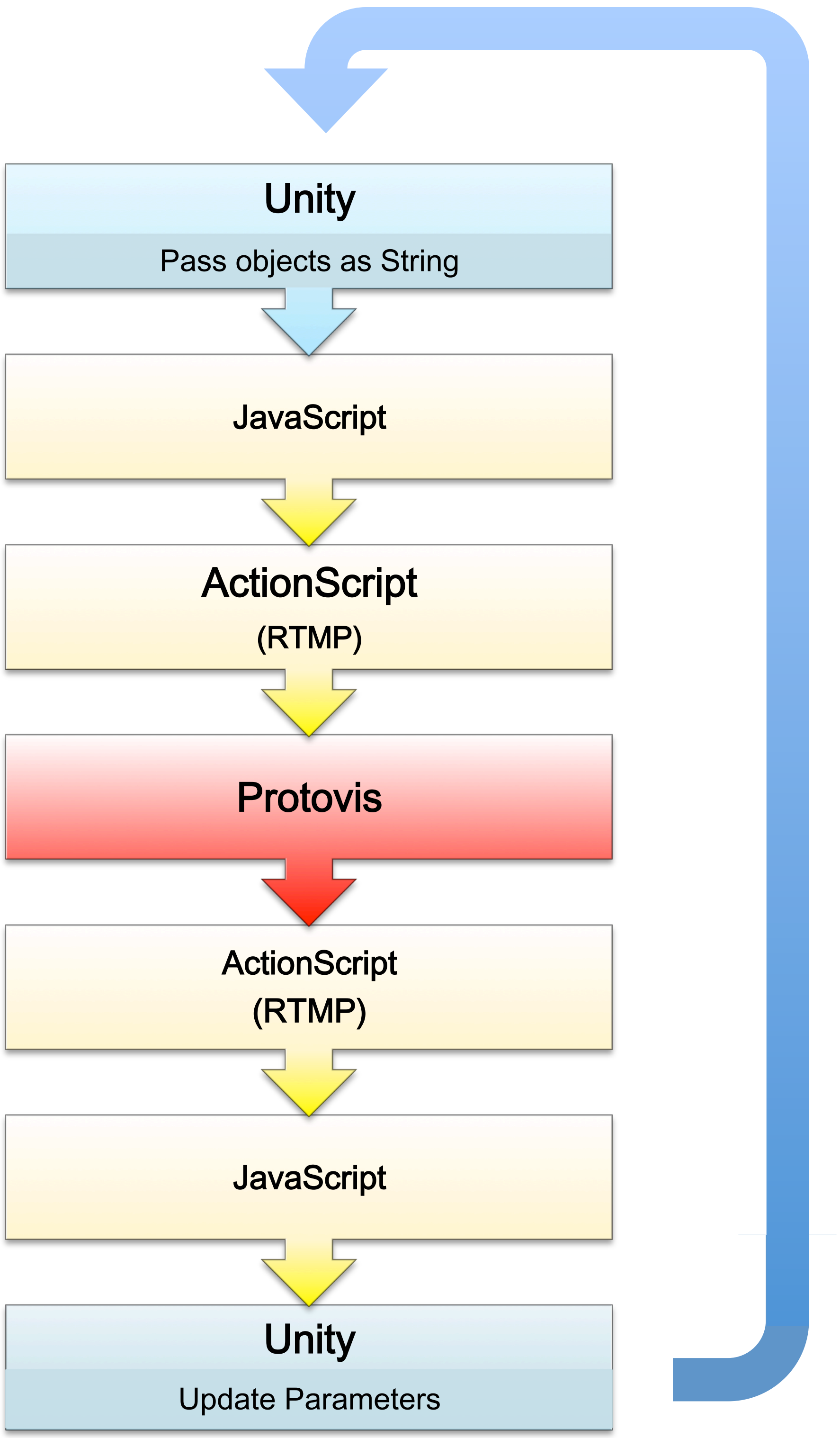


Figure 19 –Stages of VGT implementation showing the interaction between game (blue), middleware (yellow), and graph (red) components.

5.1. Integration Depending on Reflection

The *game* client's need for a generic means of enumerating objects and their fields causes VGT to depend on programming languages that support computational reflection (Cantwell Smith, 1982). Computational reflection exposes data type information for all variables, allowing them to be queried in a general way. Languages that support this type of construct include both JavaScript and C#, both of which the Unity3D game engine supports. The advantage in relying upon object reflection rests in the fact that a developer wishing to use VGT only needs to attach a script or plugin to a game's runtime, without requiring much other implementation.

VGT uses a script that will query the game for objects with fields fitting some criteria, such as being a writable number. This script then packages this information into update messages and shares them with other VGT components. These messages are compiled on regular intervals for user-defined game objects and are sent to the *vis* client for further analysis. The *vis* client then parses this information and updates its display accordingly. Below is a sample diagram of an update message from *Drift!* used to inform the *vis* client about the state of variables.

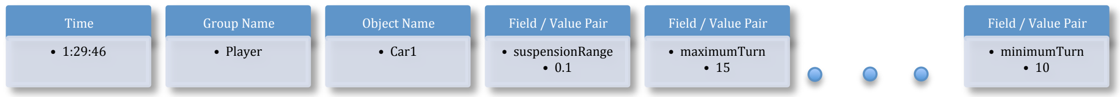


Figure 20 - Format of VGT *vis* client update protocol

The *game* client receives update messages from the *vis* client in a similar fashion. When received, a script will parse an update message and refresh the values of any objected specified in the message. This requires the *game* client's cooperation. Below is a diagram of an update message for *Drift!* used to inform the *game* of changes to its variables:

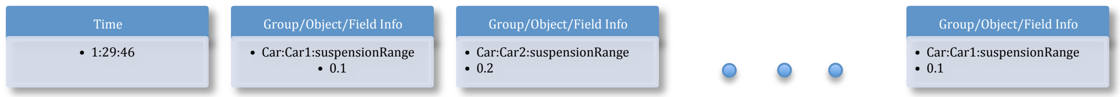


Figure 9 - Format of VGT *game* client update protocol

5.2. Interactive Frontend

Two parts comprise VGT's interactive frontend: the *vis* and *game* clients. The *vis* client is written in JavaScript using the Protovis library, and is where all of the analysis and modification of the game state takes place. The web-based Unity3D game client serves as a rudimentary feedback interface for game testers wishing to share comments, bug and preference data. An overview of each is provided with context about how they fit into the VGT framework.

5.2.1. Protovis as Visualization

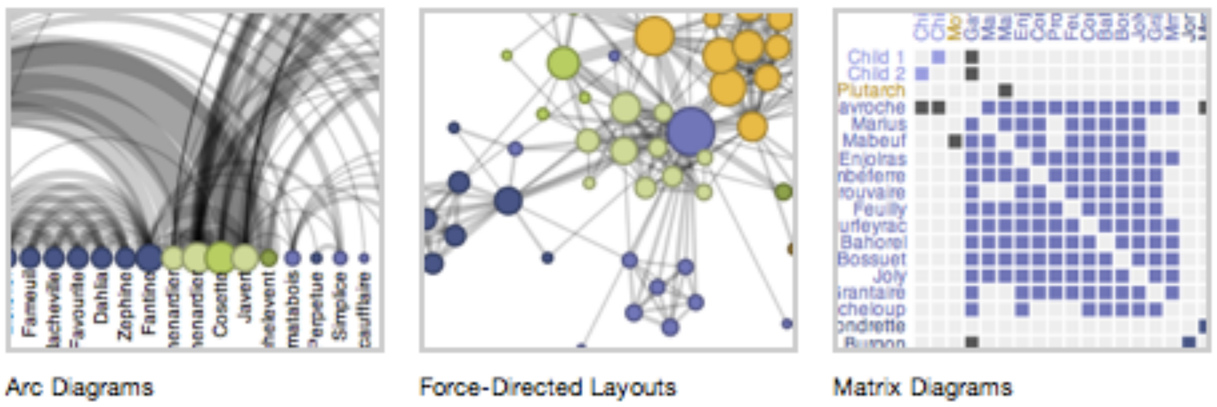


Figure 21 - Sample set of Protovis' graphical capabilities

As described by the developers of Protovis, "Protovis composes custom views of data with simple marks such as bars and dots. Unlike low-level graphics libraries that quickly become tedious for visualization, Protovis defines marks through dynamic properties that encode data, allowing inheritance, scales and layouts to simplify construction (Stanford Visualization Group)." In addition, Protovis uses a declarative syntax that helps abstract implementation details. As a result of using protovis, the full implementation of VGT's frontend was condensed into well under 1,000 lines of code.

5.2.2. Unity3D as Game

Unity3D is a cross-platform, high-performance game engine and toolkit (Unity3D Development Team). Its rich feature set includes an integrated editor, live debugger, performance monitor and an extensible UI. The primary motivation behind using Unity3D was its robust support for high-level scripting languages and web technology. Because of this, exposing Unity3D game parameters to third-party clients requires much less infrastructure than would be necessary if using other languages that lack high-level abstraction. Generally speaking, one can easily interface VGT to a Unity3D game using a couple of scripts that monitor and update the game state accordingly. Implementing the game as a web-based plugin has several advantages such as being able to rapidly define new user interactions for preference sharing as was described in Section 4.2. A higher performance alternative to this implementation would be to build a native plugin that acts as the only intermediary between the VGT *game* and *vis* clients. As a proof of concept, the implementation described here follows the former approach.

5.2.3. ActionScript as Middleware

In order to avoid the need for the creation of more-complex systems, the web-based language, ActionScript provided an excellent intermediary between the *vis* and *game* clients. Using ActionScript as the middleware allowed unique and persistent peer-to-peer connections between multiple clients to be easily established, using the Real Time Messaging Protocol (RTMP) designed by Adobe Systems Incorporated (Adobe Systems Incorporated, 2009). Using this approach makes collaborating between multiple clients seamless, as RTMP provides each with a unique key that enables both individual and global messages to be received by the appropriate party. In future revisions of VGT, establishing an intermediary server to intercept these messages may prove to be useful for more-robust analysis and logging.

6. *Drift!* Revisited – Making The Case for VGT

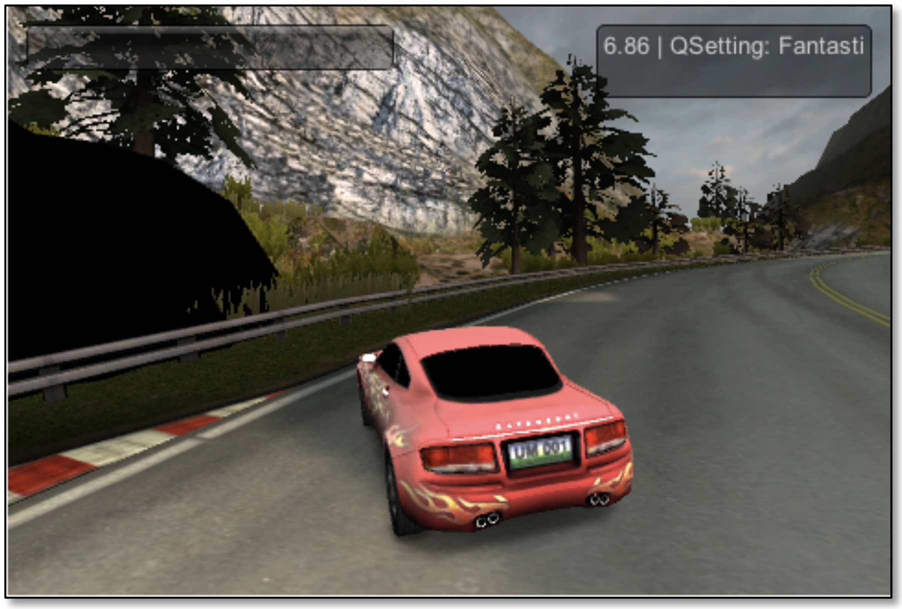


Figure 22 - *Drift!* - racing game shown with values unmodified by VGT

Here we revisit the notional game *Drift!* first presented in Section 1.2 where a developer, QA team, and subject matter expert are assessing the drivability of a simulated 1992 Mazda RX-7. In the original example, the developer made his best guess and waited for a response from the QA staff. The QA staff then worked with a subject matter expert to assess their accuracy. The QA team then logged this information in their bug-tracking database for the developer to address when he can allocate enough time to fix the issue. This cycle of game tuning is repeated until the correct values are found.

In the previous approach, the potential solutions provided were both labor intensive and time-consuming. VGT offers an alternative lightweight solution that involves relatively little setup, and can integrate easily with most modern game environments. Below follows the development and QA parties in approaching this scenario using VGT.

In Figure 22, we see an early development build *Drift!* provided by the developer. At this stage, the QA team is once again working with a subject matter expert with, the exception that they now will use VGT to assist them. As they test *Drift!,* they observed that the vehicle shown has poor handling capabilities because it turns too slowly on corners – causing the player to crash into road barriers. According to the subject matter expert working with the QA team, this does not reflect the handling of the Mazda RX-7, a high-performance sports car. To rectify this, the QA team does not have to immediately put in a report describing the behavior, but will start customizing the values in question.

To begin, the QA team will play and observe game behaviors just as before, but now an additional member of the staff will stand on the side just observing; using VGT to analyze what is going on in the game. The subject matter expert tells the testers that the car doesn't turn well enough, and that it should be more sensitive. The VGT user takes this into account and selects the maximum and minimum turn parameters for the players in question, as shown in Figure 23.

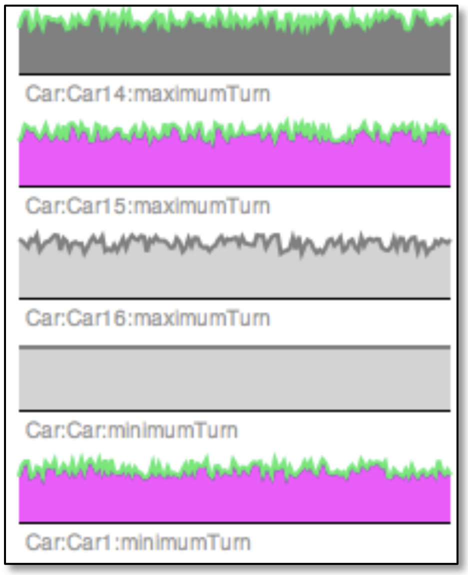


Figure 23 - Selected maximum and minimum turning fields for game example

The VGT user now has a display of all involved players and the values of their turning radius fields over time. He took note of the specific time that the subject matter expert observed that the turning radius was off, and investigates around this timeframe via the VGT Zoom & Focus panel, shown in Figure 24.

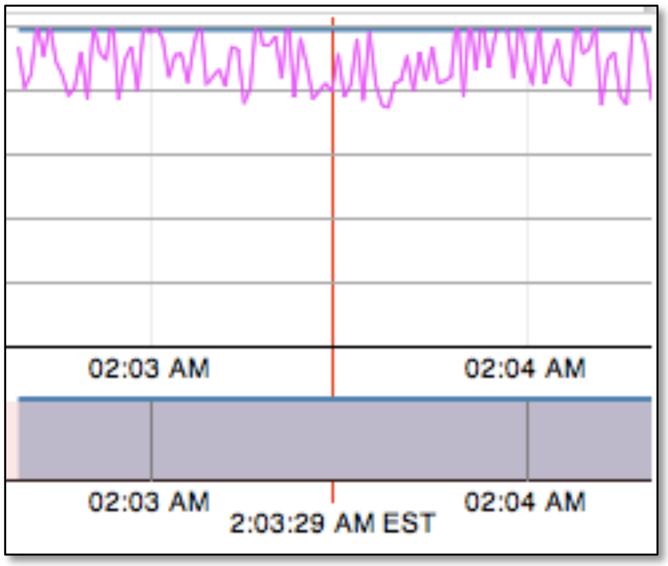


Figure 24 - maximum and minimum turning radius values over time for game example

While focused in on this timeframe, the VGT user notices that the values are varying too narrowly, and so he decides to investigate the systemic effects of modifying either of the values. To accomplish this, he then draws his attention to the Detail panel, which currently displays the value information for the maximum and minimum turning fields for all players at the time interval selected via the red index line in Figure 24. The Detail focus is shown in the following Figure 25.

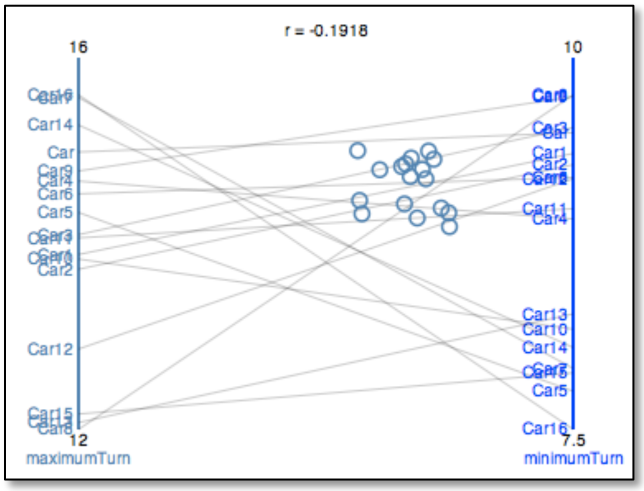


Figure 25 - maximum and minimum turning values for a given time interval in the game example

The VGT user notices that the maximum and minimum turning values are within a relatively close range, their domains ultimately differ, as is denoted by the maximum and minimum observed values. Furthermore, the Pearson Coefficient denoted by *r*, shows that there is no linear relationship between the two because it is not lower than our defined threshold of 0.05. The VGT user then goes through this evaluation process for any other values that he sees as relevant. After this analysis, he then concludes that it is safe to change individual player values for maximum and minimum turning fields, because they are independent of any other game fields and only affect themselves. Next, he moves onto increasing the sensitivity range for some players and observing its effects on playability, as shown in the Update panel in Figure 26.

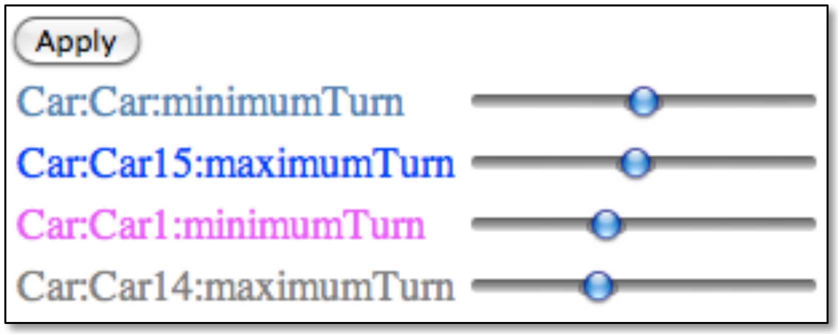


Figure 26 - maximum and minimum turning values for a given time interval in the game example

Both the QA team and the subject matter expert observe the results of such changes and will continually repeat this process until the desired game behaviors are achieved. When done with this cycle, the VGT user then wants to log the current items in focus for later analysis. This is accomplished via the Group panel shown in Figure 27.

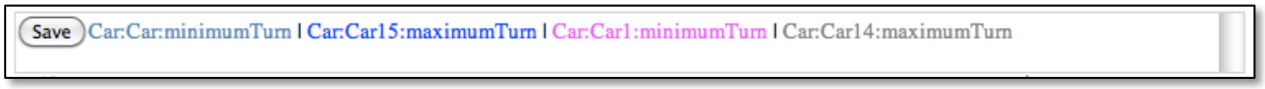

Save
Car:Car:minimumTurn | Car:Car15:maximumTurn | Car:Car1:minimumTurn | Car:Car14:maximumTurn

Figure 27 - maximum and minimum turning field for the selected players

When both the QA team and the subject matter expert are satisfied with the performance of the vehicles, the QA team will log the results just as they did in the previous scenario where they did not have the assistance of VGT. Using this information, the developers can now make more-informed decisions about which parameters will affect the player's experience, preference, and expectations.

Aside from detecting the bugs and game behaviors in *Drift!*, VGT enables the players to give additional feedback. While the primary VGT user was focused mostly on correcting vehicle steering parameters, players were busy driving and compiling their own bug, comment, and preference submissions. Doing so via the VGT game interface was very easy for them because of its integrated interface. Prior to playing, they were instructed to press the "Like" button every time they felt compelled to express their in-game enjoyment, and to press the "Dislike" button every time they observed something they felt negatively about. Some players provided additional details via the optional comment and bug submission box. Use of these controls resulted in the VGT user seeing where the players reported the most bugs, enjoyment, and dissatisfaction. Now armed with a deeper understanding of what the players wanted most, the VGT user notes these observations in his reporting of the maximum and minimum turn fields, so that the developer can make the best assessment about what should change. The indicators used in this scenario are the same used in Section 4.1.5 in Figure 14.

6.1. Conclusion

The goals of visual game tuning are to provide an intuitive means for the designers of games and other interactive media to configure their products in a manner that promotes informative collaboration. Through advancements in user experience heuristics and information visualization, there is the potential that we will realize a more generalized approach for visually tuning interactive games.

6.2. Future work

Interesting challenges lie ahead in designing a general VGT infrastructure that can support multi-modal visually collaborative workflows in a common operating environment. At present, the majority of VGT's computational analysis is done on the clients themselves, which can lead to poor performance when many objects are being tracked simultaneously. Early in VGT's design there was an expressed desire to support mobile platforms because of their collaborative capabilities; however, due to the current limitations on mobile performance and VGT's peer-to-peer infrastructure, this wasn't feasible to implement. Beyond these shortcomings, and perhaps more importantly, robust collaboration solutions don't require that collaborators are local to one another – something VGT wasn't originally intended to address as it was developed with small co-located groups in mind.

Some individuals reviewing VGT have suggested that there is potential for its use to negatively impact QA team performance by placing an additional burden on their workload. This could potentially increase cost if not addressed adequately. While there is more initial overhead, it has yet to be seen whether or not VGT will yield better overall testing performance. Though informal trial runs showed that VGT sped up the debugging process in a small testing environment.

Such issues ultimately limit the impact that Visual Game Tuning can have for practical applications until further development and testing is performed. To overcome these limitations, more robust middleware solutions must be explored. Development of a database server as the intermediary between the *vis* and *game* client will be critical in this endeavor. Doing so could have great potential to keep clients lightweight and provide richer computational analysis. Logging of high-bandwidth multimedia assets then becomes feasible, and makes adding new layers of dynamic information possible. Having the ability to stream player video captured alongside temporal game-state and preference data has the potential to be very useful in identifying the impetus behind player choices and preference. Through tightly integrated game and interactive visualization technologies, there is a great potential to completely revolutionize the way we analyze interactive media and its user studies.

Bibliography

1. Unity3D Development Team. (n.d.). *Unity3D*. Retrieved December 1, 2010, from http://unity3d.com/
2. Wilcox, R. R. (2005). *Introduction to robust estimation and hypothesis testing.* Academic Press.
3. Yuan, X., Guo, P., Xiao, H., Zhou, H., & Qu, H. (2009). Scattering Points in Parallel Coordinates. *IEEE Transactions on Visualization and Computer Graphics , 15* (6), 1001-1008.
4. Adobe Systems Incorporated. (2009). *Real Time Messaging Chunk Stream Protocol version 1.0.* Specification.
5. Andrienko, G. A. (2001). Constructing Parallel Coordinates Plot for Problem Solving. *Proceedings Smart Graphics* (pp. 9-14). New York: ACM Press.
6. Bates, B. (2004). *Game Design* (2nd Edition ed.). Thomson Course Technology.
7. Bethke, E. (2003). *Game Development and Production.* Wordware Publishing, Inc.
8. Cantwell Smith, B. (1982). *Procedural Reflection in Programming Languages.* PhD, Massachusetts Institute of Technology, Electrical Engineering and Computer Science.
9. Card, S. K., Mackinlay, J. D., & Shneiderman, B. (1999). *Readings in Information Visualization: Using Vision to Think.* San Diego, CA: Academic Press.
10. Crossley, R. (2010, January). *Average dev costs as high as $28m*. Retrieved from Develop Online: http://www.develop-online.net/news/33625/Study-Average-dev-cost-as-high-as-28m
11. Desurvire, H., Caplan, M., & Toth, J. A. (2004). Using Heuristics to Evaluate the Playability of Games. *CHI Conference on Human factors in Computing Systems* , 1512.
12. Federoff, M. (2003, June). User Testing for Games: Getting Better Data Earlier. *Game Developer Magazine* .
13. Federoff, M. (2002, December). Heuristics and Usabiilty Guidelines for the Creation and Evaluation of FUN in Video Games. *Thesis at the University Graduate School of Indiana University* .
14. Hinum, K. (2006). *Gravi++ An Interactive Information Visualization for High Dimensional, Temporal Data.* PhD, Vienna University of Technology.
15. Johansson, S., & Johansson, J. (2009). Interactive Dimensionality Reduction Through User-defined Combinations of Quality Metrics. *Visualization and Computer Graphics, IEEE Transactions on , 15* (6), 993-1000.
16. Kaner, C., Bach, J., & Pettichord, B. (2001). *Lessons Learned in Software Testing: A Context-Driven Approach.* Wiley.
17. Malone, T. (1982). Heuristics for designing enjoyable user interfaces: Lessons from computer games. In J. C. Thomas, & M. Schneider (Eds.), *Hman Factors in Computing Systems.* Norwood, NJ: Ablex Publishing Corporation.
18. Microsoft. *C# Language Specification Version 3.0.* Specification.
19. Shneiderman, B. (1996). *The Eyes Have Ie: A Task by Data Type.*
20. Standord Visualization Group. (n.d.). *A Graphical Approach to Visualization*. Retrieved December 1, 2010, from Protovis: http://vis.stanford.edu/protovis/

21. Rubey, R., Browning, L., Roberts, A., & SofTech Inc., F. O. (1989). Cost effectiveness of software quality assurance. *Proceedings of the IEEE 1989 National Aerospace and Electronics Conference. 4*, pp. 1614-1620. Dayton, OH: IEEE.
22. Tufte, E. (2001, April). *Ask E.T.: Graphing Software*. Retrieved December 1, 2010, from The Work of Edward Tufte: http://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=00000p
23. Tobiasz, M. A. (2010). *Lark: Using Meta-visualizations for Coordinating Collaboration.* Thesis, University of Calgary, Computer Science, Calgary, Alberta.