In summary, the two situations outlined – population sampling and dataset sampling are mathematically similar and have a similar purpose to ensure sample representativeness and coverage, but during the production of a game it is not possible to confirm with 100% certainty whether a sample is representative of the population – we can only estimate representativeness using statistical tests.

9.3.2 Temporal Properties of Game Telemetry

Change in both the population of a video game as well as the game itself is another factor that must be recognized when conducting sampling. Video games with heavy multiplayer components are most prone to this problem as they will undergo the largest amount of changes in their lifetime causing some data from earlier points in time to become stale and uninteresting.

In addition, many computer games, particularly single-player games, have a power-log distribution of players with a majority of the players only playing for the first few days or weeks; if actual players are being sampled this means that the players who are still playing 3–12 months after their first session could exhibit drastically different behavior than those playing for 2 days only. In both of these situations, care has to be taken in which player's data to include in the sample as a pure random sample may be systematically biased towards or away from any particular group of players, e.g. hardcore players.

9.3.3 Stakeholders

Another aspect to consider in relation to sampling strategies is the involved stakeholders. In any medium to large sized development house, there can be many stakeholders as discussed in Chap. 3. Such stakeholders are: VPs, Marketing, Level/Playfield designers, System/Gameplay designers, Programmers, and so forth. Stakeholders have drastically different needs in terms of what data to collect and how they are to be analyzed and fielded internally. As mentioned by Zoeller (see Chap. 7) (Zoeller 2011), when discussing the *Skynet* metrics suite used by him at Bioware, a VP is usually only interested in high-level statistics (How stable is the game? How many players are playing today?), Marketing may be more interested in knowing what people enjoy (combat, story, etc.), Designers are more interested in where players go and where they have problems so they can improve the game, and lastly Programmers are interested in system level statistics such as framerate, hardware platforms used and memory usage (see Chap. 3 for further discussion about stakeholders in game analytics, and Chap. 7 for more on *SkyNet*). The varied nature of stakeholder requirements directly encourages a situation where companies try to track as much as possible from every player in the population. A great number of the questions posed by stakeholders, notably at the strategic levels, can be answered via telemetry.

9.4 Sampling Strategies

There are a variety of ways in which a population or dataset can be sampled. Some approaches lend themselves more readily to population sampling or vice-versa. In this section a couple of the most important sampling techniques, for dealing with game telemetry data, are outlined. We also discuss case examples to show how such techniques can be applied. For more on sampling techniques, see e.g. Fowler (2001) for surveys or Han et al. (2005).

The major groups of sampling techniques are **probability** and **non-probability** sampling. The former includes some form of random selection when choosing the data points for the sample, and allows the calculation of statistical chance that the sample is representative of the entire dataset. In non-probability sampling, the data points sampled are chosen non-randomly, and it is therefore less likely that these types of samples are representative of the dataset they are drawn from; however, even while this problem exists, non-probability sampling is useful in specific circumstances.

Different sampling techniques have different pros and cons, and are usually suited for specific situations. An exception may be random sampling, which generally is perceived as the default sampling technique when dealing with quantitative data. However, ensuring randomness can be difficult, especially for multi-modal datasets, and sometimes it makes more sense to tailor the sampling technique to the data. For example, if investigating the behavior of players from four different pre-defined categories (e.g. following a classification analysis); or when performing A/B-testing, stratified sampling is the optimal approach to ensure each category is equally represented irrespective of the percentage distribution of the categories in the parent population.

Sampling techniques applied to game telemetry data is, as mentioned in the introduction, something that is rarely reported on. Even within academic research, very few of the published works discuss how samples of telemetry data were obtained or the criteria imposed on sampling (Chittaro et al. 2006; Gudmundsson 2009; Kim et al. 2008; Thawonmas and Iizuka 2008). However, some publications have included details of the sampling techniques used (e.g. Drachen et al. 2009; Gagné et al. 2011; Drachen and Canossa 2011).

To demonstrate the various sampling techniques, a generated dataset is used. The script for generating and sampling the data was written in Python 3.1, graphs made in R (a highly flexible tool for data analysis). The dataset describes a population of one million data points generated following the log-normal distribution, with a mean of 5 and sigma of 0.7 (see Fig. 9.2). The choice of the log-normal distribution was made, because it is commonly found in game telemetry data dealing with players and time. For example, how long it takes players to get through a particular level. There is a time at which most players finish the level (the median), a minimum time it takes to finish the level (how long does it take to walk or crawl it), and lastly there is a long tail in the distribution of people who got very lost, left the console/PC on while away over the weekend, or some other reason entirely (e.g. buggy telemetry logging code).

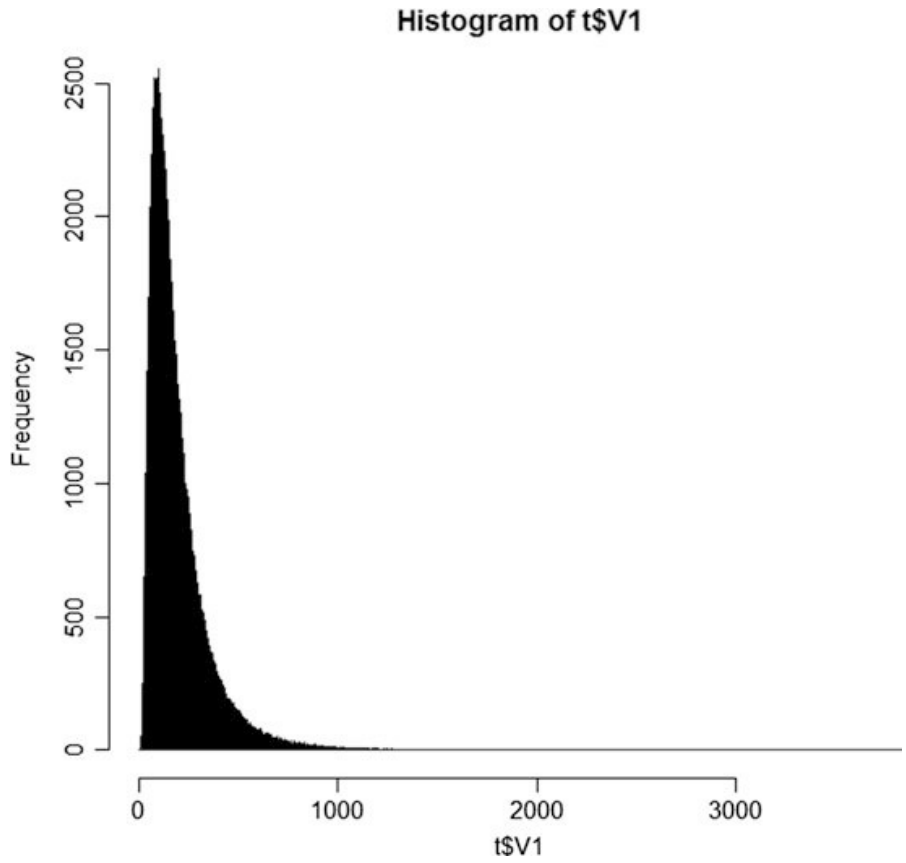

Fig. 9.2 A generated log-normal distribution of the game completion time of one million players from a fictive game

9.4.1 Random Sampling

Random sampling is the simplest and most widely used technique for sampling game telemetry data (that we know of), but also one that is dangerous to use uncritically, especially when there are structures inherent in the data. Even a truly random sample does not guarantee that all values represented will occur in the sample, and runs the risk of missing entire small clusters (Cochran 1977). Additionally, determining what random sampling is in specific contexts is challenging.

Using the random sampling strategy, a number of items (or data points) are randomly sampled from the population or dataset (the two terms are used interchangeably here). If a truly random generator is used to determine if an item is included, then the resulting sample should reflect the population to a strong degree.

There are different forms of random sampling that subtly apply bias to the results. For example, all of the items could be numbered and then items randomly chosen, either allowing **duplicates** or not, or each item could be included with a **set – or fixed – probability**. Allowing duplicate items into a sample may be desired if a truly random sample is required but they will, obviously, add more weight to any statistical values calculated based on the sample if they by chance are selected more than once (this can be problematic if outliers are selected multiple times).

Not allowing duplicate items in a random sample requires a 'bag picking' randomness in which every item is ensured uniqueness, but there could be a very subtle bias as a result of throwing away results from a random number generator (which includes duplicate values). Given a large enough population, these two sampling methods will provide similar results, however, if we sample more than 10% of the population (often happens with analysis of telemetry data of data from user testing where the population can be limited), the difference between the conclusions obtained from the two forms of random sampling may be significant. Applying a fixed probability to every item in a population to determine its inclusion in a sample means that the sample will not have a fixed size between multiple runs (even random number generators have some variance (Cochran 1977)) and different populations (which is important in post-release telemetry gathering as the population is constantly growing – so successive samples drawn using fixed probability will be progressively larger).

In our experience, questions relating to the entire dataset or population are best sampled using simple random sampling. Questions such as 'How many players played last week/month?' or 'What percentage of players starting this mission completes it?' are good examples. When dealing with continuously incoming or time-dependent data, e.g. from an MMO beta test, samples should be discarded and redrawn in order to make sure the data being worked with represent the newest build of the game.

9.4.2 Systematic Sampling

Systematic sampling is a variation on random sampling where instead of a purely random set of items (data points) being chosen for the sample, the items chosen are instead chosen on an interval from the initial population (**N**). The approach runs as follows: determine the desired sample size, and then list all of the items of the population. Then determine a random starting point and then choose every **i**th item in **N**, where the interval **i** = (population/sample size) (Sudman 1976). For example, if a population consisted of 10,000 items but the sample size is 1,500 the sampling interval (**i**) would be: 10,000/1,500 = 6.67, which could be rounded up to 7. With for example a random starting location at 4,379; we would choose the 4,379th item of the population and then every 7th from that point, until 1,500 points had been selected (Fig. 9.3).

Systematic sampling works in either the pre- or post- production phases of video game development; either the database of telemetry or the list of potential participants could be sampled in this manner. Systematic sampling is easier to use than a purely random sample when working with physically printed lists. The major weakness to systematic sampling is if there is some inherent pattern in the way the population is listed (i.e., if even items = male and odd = female (Sudman 1976)), this will bias results. However, if no such pattern exists then the sample can generally be treated as being random. Given this weakness and the digitization of nearly every list of a population or dataset in game development, purely random samples are more common.

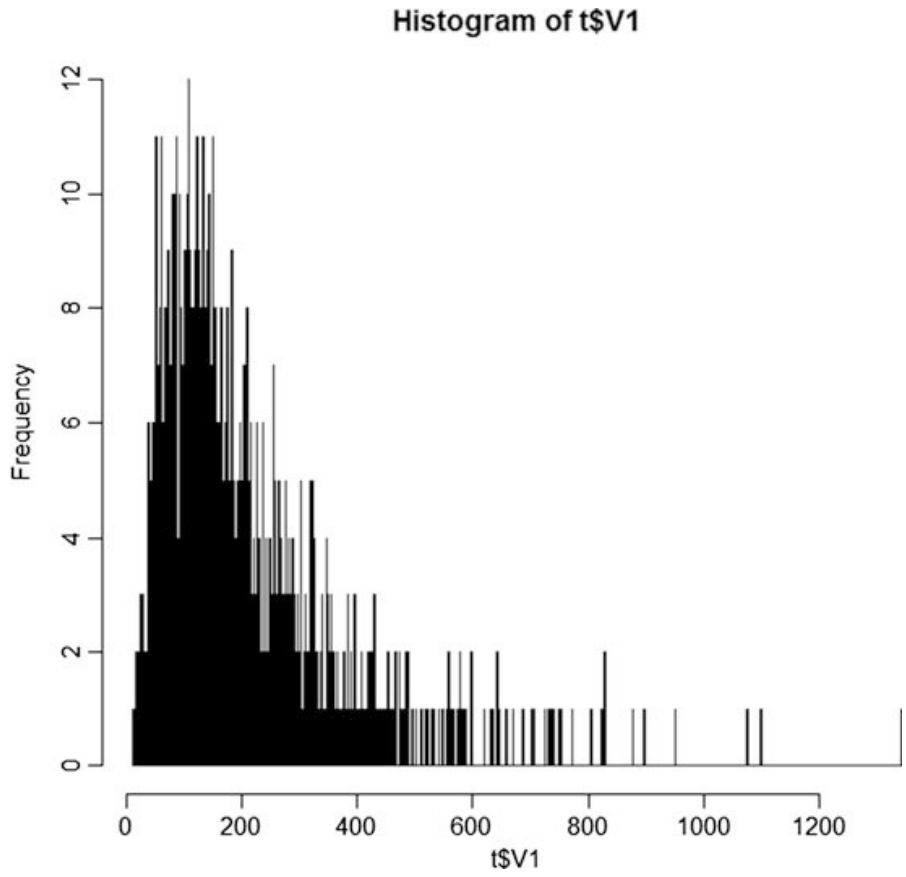

Fig. 9.3 A systematically chosen sample taking every 667th item (data set size/sample size = 666.66), visualized with 1,000 bins. The distribution is similar to the random sample of 1,500 items in Fig. 9.1

9.4.3 Cluster and Stratified Sampling

The above sampling strategies assumes that each player being sampled is part of a single population. Sometimes there are natural groups or patterns in telemetry data, however, and in these situations adopting sampling strategies that account for these patterns is useful.

9.4.3.1 Cluster Sampling

This is a technique by which the population or dataset is broken into mutually exclusive clusters; sampling is then performed from each cluster and then reassembled into one sample hopefully representing the variance between and within the clusters. For example, keeping telemetry data of a subset of players (but keeping all of it) is a form of cluster sampling. If any filtering of telemetry is occurring then clustering is probably already being used; for example, if a random amount of multiplayer matches are being recorded then clustering of events such as players who played a particular map or death events for the map are already being sampled using clustering.

Cluster sampling can be used in both pre and post production analysis. For example, pre-production player studies can try to take all potential participants from various geographic locations or contexts (e.g. guilds) but not others, this would allow for comparison across seemingly homogenous clusters. Cluster sampling based upon player ID is a common post-production sampling technique.

9.4.3.2 Stratification Sampling

This is a sampling technique that breaks the population into subgroups based upon specific variables of interest (such as age, geographic location, etc.) and then an equal proportion of samples from each group are randomly sampled. The strength of stratification sampling over a pure random sample is when trying to look at uncommon behaviors, (i.e. matches for a little played map) a representative sample for that behavior is sampled. How to break up (stratify) the population is the hardest question when dealing with stratification sampling as it requires the understanding of the population or dataset, and as a result data analysis is usually iterative with (more) stratification sampling coming in after some level of analysis has been conducted.

Stratification sampling can be used in both pre- and post-production of video games. In pre-production potential participants can be stratified by variables of interest (such as gender, game experience, age) before inviting them in to participate in a playtest.

9.4.4 Non-probability Sampling Methods

There are several non-probability sampling methods which are, however, of limited use because they do not allow the calculation of confidence. However, they can be useful when performing population sampling and are therefore briefly introduced here.

9.4.4.1 Quota Sampling

This is a technique that sees a population first segmented into mutually exclusive sub-groups, similar to stratification sampling. Subjective judgment is then used to select the people (or data) from each segment based on a pre-specified proportion. For example, it may be wished to sample 300 female and 600 male players. This allows for targeted sampling, but makes the sample non-random and possibly biased.

9.4.4.2 Snowball Sampling

This is an interesting sampling technique that can be used in situations where the resources available for user research are limited, or when trying to recruit participants from a difficult-to-reach population (Goodman 1961). Snowball sampling is

carried out by initially recruiting one or more members of the target audience, and asking them to name or recruit K more participants/members of the target audience. Once the desired threshold has been reached, sampling stops. This type of sampling is relevant when it comes to finding people to test a game and thus deliver telemetry data, at a very low recruitment cost. Snowball sampling is notably useful when the population that a studio is targeting is not familiar to them, for example a studio of mostly 20–30 year old males making a social game for 40+ housewives.

Quota and snowball sampling are not relevant when it comes to sampling from a database post-production, because at this point in the development cycle, the population is known and recorded. However, the methods are useful when resources are limited and accuracy is not important.

9.5 Best Practices

Rounding up, the following points summarize the key takeaways from this chapter, and comprise some of the "best practices" that we utilize for conducting sampling on game telemetry data:

- **Understand sample sizes**. Calculating the sample size needed in order to obtain a result of a desired significance, with a desired statistical power, is important to ensure reliable results. Understanding Type I and II errors, effect sizes, power and the relationship between samples and populations is the first step for people interested in working with samples.
- **Real world data can be noisy and incomplete**. Data from the real world can be very noisy and it is therefore important to make sure that sources of noise or incompleteness (such as players disconnecting part way through a game, gaps in logging, and similar sources of noise, gaps or bias) are understood, controlled and/or accounted for (see e.g. Drachen et al. 2009).
- **Check for normal distribution**. Human attributes (such as height) are normally distributed, the results of their behaviors when playing games (such as how long they took on a level) may not be. Thus, it is important to always check whether sampled data are normally distributed before running any tests or algorithms that require data to be normally distributed (e.g. a parametric test).
- **Check sample vs. dataset means**. If sampling from a database, where the population mean can be calculated, always check sample means against dataset means – if substantially different, re-sample – by sheer random chance we have sampled from only one section of the dataset. Alternatively, we may have sampled from the wrong dataset (human error happens).
- **Check for outliers**. The purpose of the analysis and the stakeholders within the company will impact whether the analysis is focused on outliers (which provide interesting stories, ideas and innovations; and are used to identify cheaters, bots etc.) or the major distribution (which provides the average actions of the player base) (Thawonmas and Iizuka 2008).

It should be noted that recent evidence from both industry and research, indicates that at least some player behaviors, for example playtime (Bauckhage et al. 2012) and purchasing behavior in Free-to-Play (F2P) games (Lim 2012) follow a power law distribution. This adds emphasis to the importance of outliers, as under a power law distribution, there will, for example, be a small amount of players who spend a lot of money purchasing items or other advantages in a F2P, to the degree where the outliers account for a large percentage of the total sales. It is getting common to hear analysts talk about "winnows", "dolphins" and "whales", the latter describing a minority of the players who are, however, highly valuable due to their spending patterns (see also Chap. 4).

- **Populations and design change**. The population of a computer game may change over time as highly dedicated fans finish the game and new ones join in. It is important to monitor the populations from time to time and determine if any new measures need to be collected or samples (and clusters, stratifications etc.) need to be redone, notably in persistent games. Changes to a game's design will change player behavior and therefore imposes the same requirement for re-sampling.
- **Use iterative sampling strategies**. When initiating a new analysis series, it is usually a good idea to be iterative in the sampling process, and analyzing larger samples (or the population at a shallow level) may lead to an understanding of important clusters or patterns in the data, that can then be sampled from.
- **Automate sampling, analysis and reporting**. Telemetry analysis takes time, and the more they can be automated the better, freeing up analysts to engage in exploratory analysis or simply be able to answer more questions from the various stakeholders. Telemetry analyses that will be run regularly during a production or post-launch, for example playtime monitoring, are ideal targets for automation, and allow for easy comparison between changes in a game (for example around a patch update) (Kim et al. 2008; Drachen and Canossa 2011).
- **Listen to instinct**. Sampling, statistical analysis and data mining has a strong human element, and, therefore, errors can occur (Han and Kamber 2006). Therefore, if a designer or other stakeholder gets a mental alert signal from a quantitative analysis, it is worth checking the result before implementing design changes. Human error happens, especially in a high-stress/limited resource environment like game development.
- **Behavioral telemetry vs. attitudinal data**. On a final note, it is important here to reiterate that telemetry collected from a video game does not include attitudinal data. From telemetry data the reasons for observed behaviors can sometimes be inferred, but not proven – to do this, telemetry data needs to be triangulated with other data, e.g. from surveys (see Chaps. 21, 22 and 23; or see: Pagulayan et al. 2003; Kuniavsky 2003; Kim et al. 2008; Isbister and Schaffer 2008; Drachen and Canossa 2011 or Lewis-Ewans 2012). Relying solely on telemetry makes it difficult to interpret the motivations for the behavior of the players, which can notably be problematic in terms of providing feedback to designers (see Chaps. 14, 21, and 22). For example, observing telemetry data that suggest a particular

way of solving a quest is very common in the player base might lead to the conclusion that the quest provides an enjoyable experience – but this is not a valid conclusion to make, only to infer. There can be other reasons why the quest is frequently selected – e.g. precursor for an enjoyable quest which the players know about.

9.6 Conclusion and Next Steps

In this chapter, we provided a brief introduction to sampling in the context of game user research, with an emphasis on behavioral telemetry, and with consideration as to the context sampling takes place in, and the practical process. The basic statistical considerations, such as determining sample size, have been outlined, and various best practice advice provided. However, sampling is a big topic and certainly not covered 100% in this chapter. Readers new to analytics and sampling are encouraged to investigate statistics textbooks like (Field 2005), data mining textbooks like (Han and Kamber 2006) or for survey work books, like (Fowler 2001; Lohr 2009), before making important design decisions based on sampled data (or telemetry data analysis in general).

It can be a daunting task, at times, to determine what to sample and how to sample it, but the rewards from game analytics can be great. Sampling saves resources in game telemetry analysis, and furthermore allows for rapid iteration of analyses. Correct sampling of the relevant population can give fairly accurate to extremely accurate (depending on the methodology used) reflections of the overall datasets or populations while taking a comparatively small amount of space and resources to analyze. There are several types of sampling that can be conducted depending on the amount of analysis that has already been conducted and the point in development the game is in.

These are the fundamental incentives for adopting a sampling strategy; however, sampling requires a minimum of statistical knowledge and some information about the structure of the data being worked with. In this chapter we have hopefully provided a starting point for obtaining the insights necessary.

About the Authors

Anders Drachen, Ph.D. is a veteran Data Scientist, currently operating as Lead Game Analyst for Game Analytics (www.gameanalytics.com). He is also affiliated with the PLAIT Lab at Northeastern University (USA) and Aalborg University (Denmark) as an Associate Professor, and sometimes takes on independent consulting jobs. His work in the game industry as well as in data and game science is focused on game analytics, business intelligence for games, game data mining, game user experience, industry economics, business development and game user

research. His research and professional work is carried out in collaboration with companies spanning the industry, from big publishers to indies. He writes about analytics for game development on *blog.gameanalytics.com*, and about game- and data science in general on *www.andersdrachen.wordpress.com*. His writings can also be found on the pages of Game Developer Magazine and Gamasutra.com.

André Gagné is a game user researcher at THQ. He received his Bachelor's degree from UBC, Computer Science and his master's from Simon Fraser University. His Master's work investigates the analysis of player progression.

Magy Seif El-Nasr, Ph.D. is an Associate Professor in the Colleges of Computer and Information Sciences and Arts, Media and Design, and the Director of Game Educational Programs and Research at Northeastern University, and she also directs the Game User Experience and Design Research Lab. Dr. Seif El-Nasr earned her Ph.D. degree from Northwestern University in Computer Science. Magy's research focuses on enhancing game designs by developing tools and methods for evaluating and adapting game experiences. Her work is internationally known and cited in several game industry books, including *Programming Believable Characters for Computer Games (Game Development Series)* and *Real-time Cinematography for Games. In addition,* she has received several best paper awards for her work. *Magy worked collaboratively with Electronic Arts, Bardel Entertainment, and Pixel Ante.*

References

Bartlett, J. E., et al. (2001, Spring). Organizational research: Determining appropriate sample size in survey research. *Information Technology, Learning, and Performance Journal, 19*(1), 43–50.

Bauckhage, C., Kerstin, C., Sifa, R., Thurau, C., Drachen, A., & Canossa, A. (2012). How players lose interest in playing a game: An empirical study based on distributions of total playing times. In *Proceedings of IEEE computational intelligence in games*, Granada, Spain.

Chittaro, L., Ranon, R., & Ieronutti, L. (2006). VU-Flow: A visualization tool for analyzing navigation in virtual environments. *IEEE Transactions on Visualization and Computer Graphics, 12*(6), 1475–1485.

Cochran, W. G. (1977). *Sampling techniques*. 3d edn. New York: Wiley.

Cohen, J. (1992). A power primer. *Psychological Bulletin, 112*(1), 155–159.

DeRosa, P. (2007). Tracking player feedback to improve game design. *Gamasutra*, 7th of August 2007, available from: http://www.gamasutra.com/view/feature/1546/tracking_player_feedback_to_.php

Drachen, A., & Canossa, A. (2011). Evaluating motion: Spatial user behavior in virtual environments. *International Journal of Arts and Technology, 4*(3), 294–314.

Drachen, A., Canossa, A., & Yannakakis, G. N. (2009). Player modeling using self-organization in Tomb Raider: Underworld. In *Proceedings of the 5th international conference on Computational Intelligence and Games* (pp. 1–8). Piscataway: IEEE Press.

Field, A. (2005). *Discovering statistics using SPSS*. London: Sage Publications Ltd.

Field, A., & Hole, G. (2003). *How to design and report experiments*. London: Sage Publications.

Fowler, F. J. (2001). *Survey research methods*. Sage Publishers.

Fullerton, T. (2008). *Game design workshop: A playcentric approach to creating innovative games*. Amsterdam: Morgan Kaufman.

Gagné, A., Seif El-Nasr, M., & Shaw, C. (2011). A deeper look at the use of telemetry for analysis of player behavior in RTS games. In *International Conference on Entertainment Computing* (ICEC 2011), Vancouver, Canada.

Goodman, L. A. (1961). Snowball sampling. *Annals of Mathematical Statistics, 32*(1), 148–170.

Gudmundsson, E. (2009). EVE Online | EVE Insider | Dev Blog. http://www.eveonline.com/devblog.asp?a=blog&bid=686

Han J., & Kamber, M. (2006). *Data mining: Concepts and techniques*. San Francisco: Morgan Kaufmann Publishers.

Han, J., Kamber, M., & Pei, J. (2005). *Data mining: Concepts and techniques* (2nd ed.). San Francisco: Morgan Kaufmann.

Isbister, K., & Schaffer, N. (2008). *Game usability: Advancing the player experience*. New York: Morgan Kaufman.

Kennerly, D. (2003). Better game design through data mining. *Gamasutra*, 15th August 2003, available from: http://www.gamasutra.com/view/feature/2816/better_game_design_through_data_.php

Kim, J. H., Gunn, D. V., Phillips, B. C., Pagulayan, R. J., & Wixon, D. (2008) Tracking real-time user experience (TRUE): A comprehensive instrumentation solution for complex systems. In: *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, CHI'08, Florence, Italy.

King, D., & Chen, S. (2009). *Metrics for social games*. Social Games Summit, San Francisco.

Kuniavsky, M. (2003). *Observing the user experience: A practitioner's guide to user research*. San Francisco: Morgan Kaufman.

Lewis-Ewans, B. (2012). Finding out what they think: A rough primer to user research, parts 1 and 2. *Gamasutra*, 24th April 2012. URL: http://www.gamasutra.com/view/feature/169069/finding_out_what_they_think_a_.php

Lim, N. (2012). Freemium games are not normal. *Gamasutra*, 26th of June 2012. URL: http://www.gamasutra.com/blogs/NickLim/20120626/173051/Freemium_games_are_not_normal.php

Lohr, S. L. (2009). *Sampling: Design and analysis* (2nd ed.). Boston: Brooks/Cole.

Pagulayan, R. J., Keeker, K., Wixon, D., Romero, R. L., & Fuller, T. (2003). User-centered design in games. In *The HCI handbook*. Mahwah: Lawrence Erlbaum Associates.

Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine Series 5*, *50*(302), 157–175.

Sudman, S. (1976). *Applied sampling*. New York: Academic.

Thawonmas, R., & Iizuka, K. (2008). Visualization of online-game players based on their action behaviors. *International Journal of Computer Games Technology, 2008*, 1–9.

Zoeller, G. (2011). *Game development telemetry*. Presentation at the Game Developers Conference 2011, San Francisco, CA.

Chapter 10
WebTics: A Web Based Telemetry and Metrics System for Small and Medium Games

Simon McCallum and Jayson Mackie

Take Away Points:

1. Description and documentation of an open-source C++/HTML-based tool for logging, storing and analyzing game telemetry for small-medium sized games.
2. Discussion of some of the key design concerns that emerged during the development of the drop-in style tool for game telemetry and metrics.

10.1 Introduction

In the 1980s game developers were able to pride themselves on developing every part of a game, sometimes even to the hardware being used. However, with the increasing size of game development teams, and the increasing demands on smaller game titles, the use of middleware and other third-party tools are becoming increasingly important to the industry. While there are still game developers who are reluctant to employ code not developed in-house, this position is getting harder to justify with the changes in the marketplace and development strategies. An example of this can be seen with the widespread use of game engines from third parties, e.g. the Unreal Engine and Unity 3D.

Metrics are an ideal candidate for a middleware tool, as discussed e.g. in the Unity interview in Chap. 8, and in Chaps. 6, 7 and 16. Until recently, very few games are sold due to their great use of metrics or the depth of gameplay analysis provided by metrics. However, this is changing with the rise of the Free-to-Play genre, where telemetry analysis forms a means of testing designs that is as important as the design itself (Fields and Cotton 2011). Additionally, game metrics analysis

S. McCallum (✉) • J. Mackie
Gjøvik University College, Gjøvik, Norway
e-mail: simon.mccallum@gmail.com

M. Seif El-Nasr et al. (eds.), *Game Analytics: Maximizing the Value of Player Data*,
DOI 10.1007/978-1-4471-4769-5_10,

can be used to make good games better. Thus, the design and development process can also be improved by integrating game telemetry analysis into the incremental development of the game. Large game developers have the financing required to develop and support an in-house metrics system. For example, Microsoft Studios Research have developed TRUE (Kim et al. 2008; Thompson 2007), and other big publishers have similar systems for collecting, storing and analyzing game telemetry and other user research data (Drachen et al. 2009). Indeed with the increase in the use of data driven game design, game data collection and analysis is becoming a larger part of many game companies' budgets (Fields and Cotton 2011).

Small to medium-size developers will however typically, due to financial reasons, need to use middleware or third-party tools rather than invest the time in developing their own solutions. In this chapter, we present the WebTics API – an API packaged as a product that is aimed at smaller developers. We will discuss the design decisions and the approach to the telemetry system that the API enables, in the hopes that developers will be able to simply import this code into their projects and improve the quality of their games through the information it provides.

Middleware tools can be incorporated into games at many levels (Isbister and Schaffer 2008; Zoeller 2011), also see Chaps. 7 and 8. In fact some game creation tools are little more than a collection of middleware, for example Delta3D (http://www.delta3d.org). Graphics engines led the way for the acceptance of middleware within the game industry. The reuse of graphics engines and game engines, like Unreal, within game companies led to the realization that the technology developed to create a game could be sold to other companies. The widespread of Unreal and Unity engines by the industry serve as a good example of this. Companies now realize the value of middleware tools.

This expansion of the use of middleware is part of the motivation for our API and tools. In this chapter, we concentrate on *WebTics*, the tool we created, specifically describing the underlying design considerations, as well as three examples of its use. We also include a documentation-style description of the API as an appendix. The chapter is divided into three general components: Sects. 10.1, 10.2, 10.3 and 10.4 is focused on a discussion of the current tools and design considerations of WebTics, Sect. 10.5 describes case studies showing various scenarios of usage (use cases), including programmers looking for bugs, designers balancing a game, and marketers/QA personnel investigating usage numbers. Finally, the Appendix contains a documentation of the WebTics API.

10.2 Types of Metrics Middleware

Game metrics middleware systems can be categorized as: products, services, or as a mixed model. **Products** are systems that you can purchase to add to a game. **Services** are provided based on a centralized-server, or in the cloud, and are categorized as Software as a Service (SaaS). In this model you do not host the software or the data, but merely interact with the system, usually through a web browser. Google Apps

(http://www.googleapps.com) are typical examples of SaaS. Mixed systems provide some tools that are independent and some that are hosted by the metrics company.

The business models for each of these types of products are different, as are the best situations to use a particular option. When evaluating the best option for a specific game development situation it is important to assess the advantages and disadvantages of the each type of middleware. Service-based metrics middleware (such as: the ROAR Engine,[1] Kontagent[2]) include:

- **Automatic updates**. Your metrics tool will always be the latest version, with no additional effort. Note that this could be bad if you are trying to directly compare results and something has changed in the tool you are using to collect data.
- **Fewer compatibility problems**. The web-based link to these services usually runs on any browser; logging calls from your game goes through a standard web interface.
- **No installs**. You do not need a DBA to run a database, network coders to manage user data, or dedicated machines to deal with the potential traffic.
- **Small initial costs**. These systems often ask for monthly fees rather than large upfront investment.

Buying a middleware metrics product, such as Playtomic[3] or Orbus Gameworks,[4] has the following advantages:

- **Control**. Having internal functionality and measurement information stored entirely within the company's intranet may be important to protect new Intellectual Property. Storing this data on a remote server can be a deal breaker for many businesses. Relying on other companies to protect and provide data is a risk.
- **Offline functionality**. In house testing behind a firewall is possible regardless of Internet connection. This may also allow external testers with poor Internet connection to test the game while offline and upload logging events when they reconnect. Only testing games when there is a good Internet connection may leave out specific bugs.
- **Single payment**. Products are purchased once and serve their purpose without usually requiring additional monthly payments. Some companies offer support and updating services for products. This starts to become part of the mixed model of product and service.

WebTics is designed to be a telemetry and metrics product rather than an ongoing service, much like Playtomic. The main difference is that WebTics is open source, and provide complete control to the developer using the system. This approach will suit companies that have the internal resources to set up a server with SQL and PHP, and an interest in being in complete control of all their data.

[1] ROAR engine. http://roarengine.com

[2] http://www.kontagent.com/

[3] https://playtomic.com/

[4] Aleph Metrics Suite from Orbus Gamesworks http://www.orbusgameworks.com/solutions.php

We do not provide a side-by-side comparison of the many game metrics systems that are currently available, as a comparison of features would become out of date very quickly.

10.3 Types of Game Metrics

Game metrics can be internal to the game, or recorded by external sources. Internal logging comes from monitoring the actions in the game, either created by the player or from the game itself. There are many types of external sources of telemetry and metrics information. Standard user testing with a trained observer provides large amounts of external information.

The focus of WebTics is to support internal logging, but it is designed to be able to integrate with external systems. The main telemetry included in the systems comes from calls embedded in the game. These internal logging calls are added by the developer during the implementation of the game, usually with a specific logging objective in mind. This type of logging will only be able to tell us about the player actions or the operations on the game client. It does not, for example, give us additional information about the player, such as the context of play, gender, distractions, accessibility issues or how the player experiences the game. External sources of information include biometrics, production metrics and performance metrics. Even within the scope of behavioral telemetry, it is difficult to incorporate different types of sources and types of game metrics data into a single unified resource.

When collecting different streams of data synchronization becomes an issue. There are various ways to engineer the environment of a game telemetry logging system to assist synchronizing different sources borrowing from synchronization methods used by other media. For example, in traditional film production a "clapper" is used to produce both a visual event and an audio event. These events are used to align the audio recording with the video recording. Another example would be using a psycho-physiological device such as electroencephalograph (EEG) which records brain activity, and linking this with a video recording of a player; a simple tap of a finger on a specific sensor will give a detectable event in both data streams (Mandryk 2008). Thus, for synchronization of game telemetry, we can use similar approaches. To assist in the synchronization of various data sources, WebTics provides microsecond timestamps for all logged events. This can be used post-session to synchronize the various sources, including situations where external data are used, e.g. from psycho-physiological sensors (Mandryk 2008).

Once we have created an environment that supports the collection of various metrics sources we can improve the usability of the data by standardizing the way in which we store the logged events. There are a set of features that are common to all events that can be logged. For events to be measurable they must be:

- **Present**: you cannot measure when something does not happen. You cannot measure the fact that there are no bugs in the system. However, you measure when errors occur, and the length of time between errors. It is important to remember to

adage "absence of evidence is not evidence of absence" (Altman and Bland 1995). This focuses the logging on things that can be measured during game play.
- **Observable**: there is no point trying to measure something that cannot be directly observed either as events in the game or by using some external sensors, such as heart rate monitors.
- **Specific**: the event must have a clear definition of when it occurs. If the intention is to measure Actions Per Minute (APM) (Lewis et al. 2011), you must define what is an action. Is each keystroke an action? If do you do for typing in a chat window? Are you interested in all clicks, or only those that relate to changes in game state?

These principles can be used to ensure that the data recorded achieves the desired outcome. Some recording objectives might require additional input external to the game actions, other may need to be described so that what is being recorded has meaning. Understanding the goal of recording data can significantly increase the likelihood of being able to use the data you have collected.

This set of features also help explain why it is difficult to request a programmer to include a metric reading of something like "fun" during game play. Fun would be *present* but it is difficult to identify how to *observe* fun, and even more difficult to describe the *specific* thresholds on sensors that indicate that the player was having fun (Yannakakis 2012).

Events can also be broken down temporally into two different types of events:

- **Instantaneous**: Events that pass a threshold value are registered with a specific time.
- **Duration-based**: Events that span a period of time and are defined by continuing action.

In our event logging system we decided to implement only instantaneous events. This is based on the principle that events that record a duration must have both a *start* and an *end* to define the duration. Therefore, we can store those *start* and *end* events and calculate duration as a feature, with post-processing and data mining. This could be achieved in WebTics by calling (see Appendix):

```
metricsSystem->LogEvent(EventTypes::DragnDrop, Event-
Subtypes::Start, m.x, m.y, 0);
metricsSystem->LogEvent(EventTypes::DragnDrop, Event-
Subtypes::End, m.x, m.y, 0);
```

The amount of data stored, and the internal logging of the metrics system have a significant impact on the sensitivity and usability of the metrics data (Kim et al. 2008). If you do not record enough detail about the actions of the player, it may not be possible to extract information about play styles, errors, or dominant strategies.

In WebTics we have decided that each event will log, not only the data provided by the programmer, but also three additional data items:

- **Time**: time since logging session started
- **User ID**: associating data with a specific instance and user of the game
- **Game Session**: for associating data with the correct session

This additional data provides the minimum set of information not only for simple aggregate-based metrics (counting events/time period), but also for more complex data mining techniques (see Chap. 12 for more on game data mining).

It is also critical for consistent evaluation of the data that events are associated with the specific **Build Number** that generated the events. In WebTics we do this by linking each Game Session with a Build Number. Build numbers are used in software engineering to specify the exact code used to create an executable. If any part of the code is changed, the build number is incremented. The current version of Microsoft Word used for this document has a build number of 120,421, suggesting that there has been over 120,000 compilations and integration of the application. If a change in the code has resulted in a change in the player interactions, each event must be able to be associated with the specific build. In WebTics the Build Number is added to the OpenMetricSession call at the start of the session:

```
metricsSystem->OpenMetricSession( uniqueID, BuildNumber );
```

With the build number associated with the recorded events we can start to perform consistent data analysis.

For the analysis of data, the initial metrics that can be easily extracted are:

- **Frequency**: how often an event is occurring. This requires that every occurrence of a specific event type be recorded.
- **Correlation**: what events consistently occur with other events. This uses the time and sequence order of the recorded events to find events that co-occur.

Correlation requires more processing and information to calculate than frequency. Correlation can be as simple as determining if event A occurs within a set time of event B. In Diablo3™ on release there was a correlation between giving a Templar follower a shield and having the game crash (Johnston 2012). This is calculated by looking at events in the moments leading up to a crash.

10.4 Design Approaches

Our approach to designing WebTics was a mix of bottom-up and top-down design. Some of the system features originated from analyzing the data that we had previously collected in an adhoc fashion in earlier games, and the others from a top down assessment of what might be needed in future. We combined events that are easy to log with ‘what if sessions’ to generate a wider array of measurements. We believe that this approach to the problem of developing a metrics system allows the developer to both focus on the low-cost high-return data, as well as directing the development toward the ideal metrics. Merely focusing on recording data would not provide room for extensions to complex interactions, while focusing only on the complex metrics would create a large barrier to starting to use the system.

During the design of the system we discussed the types of users and their objectives (Use Case analysis). It is often tempting, as a developer, to start looking at data being generated from your game and get lost in following what looks interesting at

the time. Although this "Grounded Theory" (Charmaz 2003) approach to developing an understanding of the game data as it is collected might work in an academic setting, usually in a commercial environment there isn't enough time to "muck about" playing with the system without generating actionable recommendations for the game (Pagulayan et al. 2003).

In the design of WebTics we have several constraints. The data must be human readable, encourage and follow good software engineering approaches, and be scientifically and legally sounds. The following subsections describe these constraints.

10.4.1 Human Readable Metrics

One of the potential problems for a number-based metric system, like this one, is ensuring that there is always a consistent translation from a block of numbers into human readable logic. From personal experience, a database of carefully collected metrics becomes meaningless when the link between the meaning of the numbers and the actual numbers is lost. When recording data it is far better from a storage space perspective to store just a numeric value that represents a category, rather than storing a string. Storing numeric values rather than strings requires a translation between the name of an event, for example PlayerDied, and the associated number EventType 3. This link is sometimes called a Data Dictionary (Dustin 2002). This approach to condense data has implications in terms of introduced biases, as discussed in Chap. 13.

In designing our system it was essential to have a clear way of updating and maintaining the Data Dictionary. Our solution to this problem is to use a three-part system:

1. Include a version and build number when starting the logging process.
2. Include a set of strings that are associated with the numeric event data types.
3. Create a back-end table to associate each event number stored with its name in the current version/build.

This solution allows programmers to use enumerated types for event logging. Enumerated types provide a conversion between the name of a variable and a number, e.g., in Java java.util.Calendar.TUESDAY = 3. These enumerations minimize the size of the data stored in the database and gives a unified way of updating the associations between the metrics stored and the events in the real world. The data dictionary allows the translations, so that we can understand what it all means.

10.4.2 Build Numbers

Recording the version and build number for programs is part of good software engineering (Henderson 2008). It takes on additional importance when dealing with metrics data, e.g., for A/B testing features (see Chap. 4) (Fields and Cotton 2011). If the data you collect is not clearly linked to the code that generated it, much of the

value of the data is lost. If you are using a version control system (which should really be standard for any programming project) including the version number of the repository is also valuable for tracking the source that generated the data. Other industries also have a strong focus on version and batch numbers. Medical research, for example, has a very strong requirement that every test is associated with a specific batch of medicine.

The system we have developed does not enforce any particular numbering format for versions, builds or repositories. However, the database back end supports a dot separated format "majorVersion.minorVersion.build.repositoryVersion" (for example "2.1.120.8756") to perform some additional continuity checks.

10.4.3 Privacy Requirements

One of the significant decisions to be made when logging player's behaviors is how much information you can and want to log, and to what extent that information can be traced back to an individual. In the US and various European countries privacy and data storage laws provide legal protection to your players. Any metrics middleware system has to be created with the ability to ensure that it complies with the law of as many regions as possible. However, no matter how much care is taken in the development of a telemetry and metrics system, it is still the responsibility of the users of the system, the game developers, to ensure they are operating within the current laws for their country (see references below).

In the design of our system we have included the ability to query and update an authorization table. This table contains the date and level of authorization given by the user. The table includes an expiry date for data related to that user. Rather than automatically deleting the data at that date, the system warns the developer that the data has expired and that the "flush expired data" page should be called. This removes the data from the database.

While this process may seem overly cumbersome for small developers, the legal issues related to storing data can be extremely serious, and small developers often do not have the spare cash to pay for lawyers to defend these cases. We hope to minimize the risks by providing the infrastructure for managing player authorization for data collection. In the default logging of player interaction we suggest using anonymous session IDs rather than unique player IDs unless the developer is using the authorization system.

Regional guidelines:

- For European developers the legislation is from the Directive 95/46/EC of the European Parliament. Search the Europa web site using CELEX and number 31995 L0046.
- For those in the UK, the Data Protection Act 1998 is the relevant legislation, and the Information Commissioners Office is the auditing body.

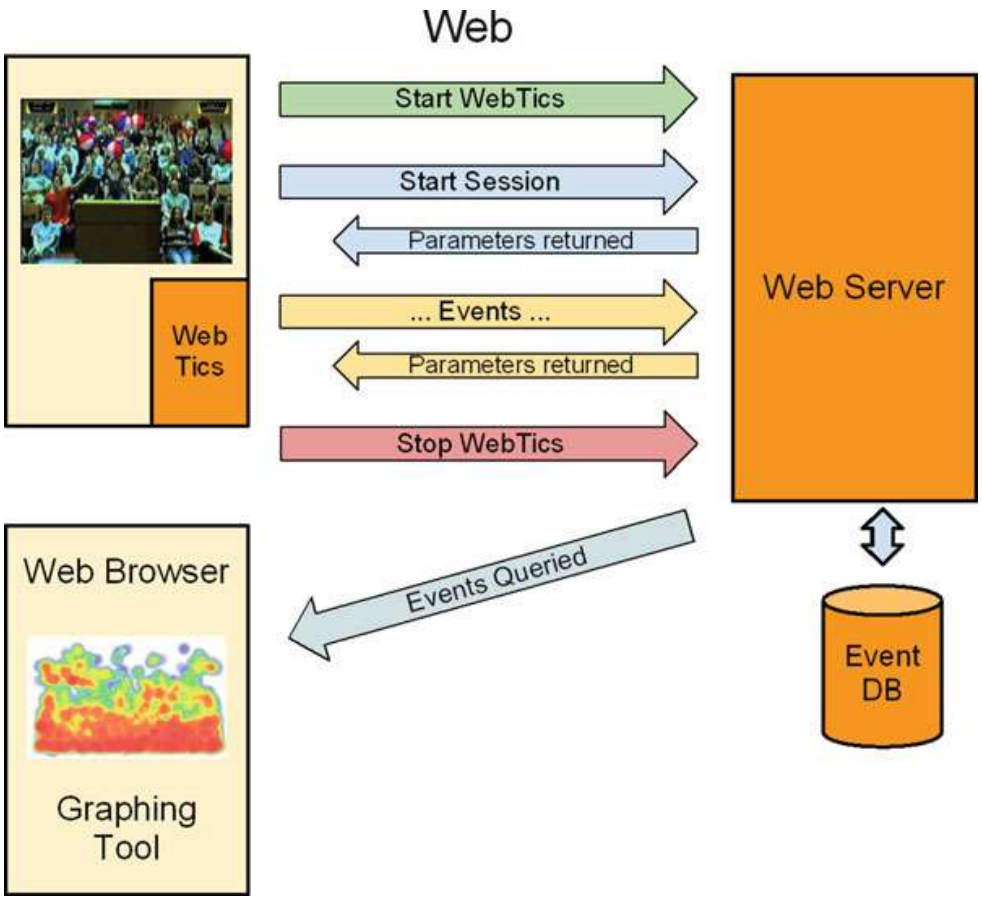

Fig. 10.1 Diagram of the components of WebTics and the interactions between these components

- In the US the FTC Fair Information Practice Principles are the guideline for storing and processing information about users that may be able to identify them.

10.4.4 System Overview

The API for WebTics is shown in the Appendix at the end of this chapter, However it is useful to have an overview of the system before we move on to discussing various case studies in the next section.

WebTics is designed as a light weight system. Figure 10.1 shows the conceptual modules of the system. The main interactions are handled by simple URL posts to the server that is running the database. These calls contain the event codes, variables, and time of each event, as well as the unique identifier for each session. Normally the API handles the session details, as this allows the programmers to focus on the metrics they are recording rather than the process of storing them. However, there is a clean separation of the database processing and the WebTics API. This allows developers to create their own versions of the API, database, or visualization system.

The workflow for a single game application is:

- Initialize the API singleton, using build number and user id.
- Start a session
- Log events
- Close the session
- Cleanup using dispose on the API logging singleton

Having implemented the logging, and run some game sessions, the database will have a large number of events. These events can be viewed directly as a database table, or using the Javascript based visualization tools.

As part of the set-up for a session the database server can be asked to return parameters. This allows dynamic adjustments to the game without having to recompile and send a new patch. This returning of parameters can also be used to generate interactive questioning systems. When events are sent to the server, there is the option for them to return data. This data could be new parameters, or questions that can be asked of the user.

10.5 Case Studies

It is often beneficial to see how a system might be used before spending the time and resources to test the system in your own environment. For details on implementation and including WebTics in a project see the Appendix at the end of this Chapter. The following section discusses three examples of the usage of WebTics. If one of the situations is close to the reader's requirements then these cases will hopefully provide a good starting point for considering the use of this metrics' middleware.

10.5.1 Examples of Using Metrics for Designers

The rise of social gaming has created a new model for game development and publishing. Rather than developing a complete product and releasing a full game, social games are usually released early as small games that are then incrementally built as the game becomes popular. This incremental improvement to the game can be informed by players' actions in the current version of the game. This data-driven approach (Rabin 2000) emphasizes the use of data to understand and improve the game, rather than merely use the intuition of the designer (Isbister and Schaffer 2008; Pagulayan et al. 2003; Derosa 2007). There are pros and cons to this approach, as it tends to develop incremental improvement rather than radical new designs. However, to ensure the best possible use of this number-based approach, it is useful to collect large amounts of data from players' interactions with the game. Simple figures such as final score or play duration may not capture the underlying player dynamics essential for improving the game, more information is usually needed.

As designers, we have used data from play-testers in various different ways. When improving a game for stadium audiences called BallBouncer (Sieber et al. 2009), we used the location of interactions to manipulate the "random" generation of balls. This improved the player experience by focusing the interaction on the areas of the stadium were there were active players. The use of player data in the development of our games can be categorized into three different types: game balance, game rule evaluation, game player surveys. These categories will be the subject of the next subsections.

10.5.1.1 Game Balance

In many games it is important that the game is well balanced to allow the player to experience the full range of activity in the game. Complex rule sets make it very difficult to predict all of the possible interactions that will occur during a play session. It is inevitable that some players will think of strategies that none of the design team anticipated. By using a metric system, a designer can identify which players win consistently, and then look at which tactics they are using. These emergent dominant strategies can then be analyzed to decide if and how they should be nerfed (have some aspect downgraded to weaken the overall strategy).

10.5.1.2 Game Rule Evaluation

It is increasingly common in free to play games to perform A/B testing on specific aspects and parameters of a game (Fields and Cotton 2011). The WebTics system incorporates the ability to have feedback from the database to the game. This can be used to update parameters in general, but also given specific user IDs it can serve a different set of parameters for different groups of users. Each user ID can then be associated with a parameter set, and the differences in critical variables, like play time and revenue per user, can be collected and compared. This eliminates the need to have different executables, and distribute them to the different user groups.

10.5.1.3 Game Player Surveys

Finally, the designer can use the in-game question system to survey the players at different stages **during** their game session. The designer can insert events that trigger questions to be sent from the server to the running game. In the game, developers have to insert some way of handling the question's XML format returned from the server. Given that the questions are tagged to an individual user it is then possible to limit the number of questions asked, or to ensure that the same questions are not asked multiple times.

These in-game surveys results will give designers different information when compared to data from the end of the play-session. There is, however, a risk of getting

poor feedback by harassing the player and breaking the flow of the game with too many questions. This feature should be used in the early testing phase rather than late in production.

10.5.2 Examples of Using Metrics for Developers

Event logs can easily be used for standard bug detection. Looking at the events that are logged immediately prior to an unexpected session end, will lead to finding the event that led to the crash. However the power of a metrics system is the ability to analyze the telemetry data, and find bugs caused by complex interactions. Rather than looking for events that lead to a crash, we are looking for combinations of events that resulted in a bug. The errors may be emergent, only becoming evident after a specific series of game events. By logging each key press or memory event, the recorded metrics may be data-mined (see Chap. 12) to establish if there is a pattern of actions that lead to a crash.

While development is in-house, this information could be logged to disk using standard logging tools. Note, however, that if data is logged in multiple systems, it may be difficult after release to collect all the data logs from the various systems. Using a system like WebTics enables the developer to assess errors on remote systems, e.g. via crash messages.

We have used the system to monitor multi-threaded programs, as these can be particularly problematic. The order of events occurring in a program with several threads often cannot be specifically predicted. The exact sequence of events across different threads may lead to either crashes or thread blocking. Logging of events that lead up to this fail state can greatly assist in tracking the cause of these bugs.

10.5.3 Examples of Using Metrics for Academic Evaluation

An important part of the games industry are games used for purposes other than entertainment, such as Games for Health (Sawyer and Smith 2008), exergaming (Göbel et al. 2010), and Game for Education (Gee 2003). The Wii Fit™, Zumba Fitness™, and BrainAge™ games are commercially successful examples of games that are designed to have effect outside of the game world. These are often collectively called Serious Games. Most of these games are trying to change the player's behavior, knowledge or attitude. To demonstrate the effectiveness of the game in changing the player it is important to provide data supporting any claims, otherwise the intervention would not be evidence based. Examples of these are also discussed in Chaps. 31 and 32 and in the interview with Serious Games's CEO Simon Egenfeldt Nielsen.

Logging of player behavior in a game can be used, not only as a way of improving the game, but also as a diagnostic tool. As part of a Masters project investigating independent play on a tablet quiz game for the elderly (Askedal 2011), it

was important to record the motor performance while playing the game. Clicking performance was logged for both correct answers and all inaccurate touches. The ratio of successful touch actions could then be compared to the number of misses using simple frequency over time. This was used to show the initial learning of the touch-based interface. The success rate in this session creates a baseline for each user. If the current performance on the game interface changes suddenly, the success metric for touch interaction could be used to indicate that there may have been significant changes in the physiological health of the player.

10.6 Visualizing Data

The human brain is amazingly powerful at finding patterns and correlations, so good that it often finds patterns that do not exist. We can, however, use this power to gain insight into the meaning of the telemetry data being collected. Any good metrics API must provide tools for standard visualizations of the data collected. The simplest form of these is the highscore list, which visualizes one dimensional (1D) data set – the metric of how many points the player has scored during the game.

Graphs and charts can be helpful when viewing larger data sets. Graphs are particularly good for spotting trends. A 2D scatter-gram places a different metric on each axis. This allows simple outliers to be identified quickly, and simple correlations between the data to be spotted. In WebTics we use the D^3 Data-Driven Documents javascript library (D3.js) (Bostock et al. 2011) for creating visualizations of recorded data. This library focuses on transforming data using web standards.

Given that most games occur in a 2D or 3D environment, and that many 3D games have 2D game play, the (x, y) location of events provides a valuable insight into the data. WebTics includes the time, and critically the location of events. Including the location allows powerful visualizations of the data. When Halo3 (Microsoft Game Studios 2003) was released, the developer, Bungie, provided heatmaps for each level showing spatial metric data, such as: where characters die, where they scored kills from, how far they got when they died, etc. (Thompson 2007). Heatmaps are relatively easy to produce once the data has been collected and organized. A human viewer can quickly identify choke points, sniper locations, and general game imbalance.

Heatmaps can be applied to almost any positional data. The concept is that events create heat at a point and that the event has an area of effect. The larger the number of events the "hotter" the area. The heatmaps in WebTics use a javascript tool developed by Patrick Wied, called *heatmap.js* (http://www.patrick-wied.at/static/heatmapjs/). This uses radial gradient functions and a customizable color gradient. The general process for producing a simple heat map image can be achieved by adding a Gaussian distribution to a point (x, y) in a 2D array. The color displayed is selected based on the value in each pixel of the accumulation storage array. The range of the data used to generate the heat map can be adjusted manually to inspect areas where there are few events, and so would have low contrast in the full heat map.

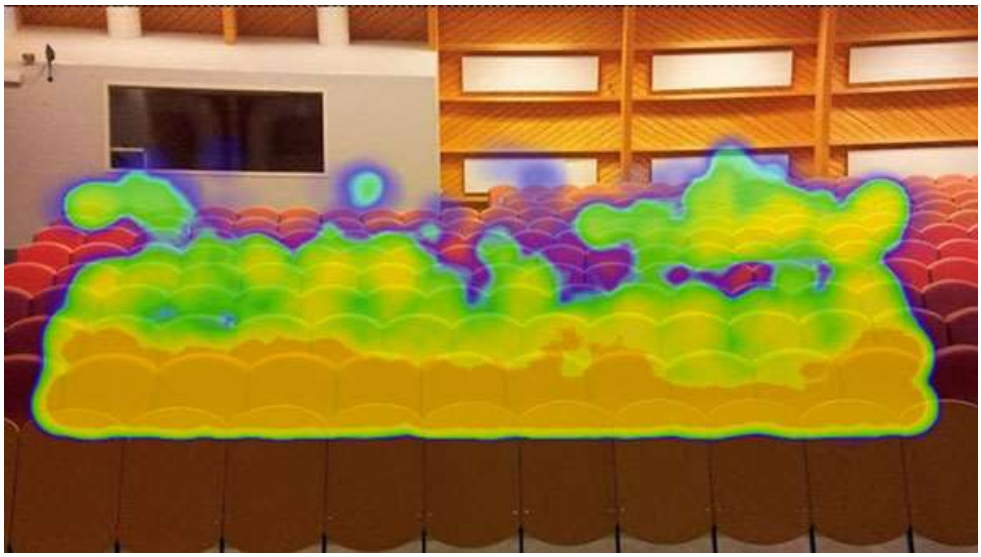

Fig. 10.2 A heatmap produced from the position of ball interactions in the game BallBouncer, seen on the auditorium in which the game was played

Figure 10.2 shows a heat map from the Augmented Reality game BallBouncer (Sieber et al. 2009) used for stadium entertainment. The game logs every successful interaction with the virtual items. From this map it is easy to see that most of the activity is happening near the front of the auditorium. This allows us to adjust the parameters in the game to improve the distribution of interactions.

10.6.1 Data-Mining

The WebTics system provides data visualization tools and very simple data mining tools. At the time of writing, the data mining tools only include frequency and binary correlation analysis. This allows the developer to analyze the frequency of the events that are logged to the system, and find correlations between selected events. This very simple analysis is performed on the server side using PHP. For more complex data mining (see Chap. 12) it is possible to use either the Knime tool initially developed at the University of Konstanz, Germany (Berthold et al. 2008), or for those who are more interested in the low level machine learning and programming analysis the WEKA system (Hall et al. 2009).

10.6.2 Extensions to the API

During in-house testing and academic research it is often useful to have two different computers linked to the same play session. One computer playing the game

while the other computer is recording video and audio from the play session. While it is possible to have two computers with synchronized clocks, the focus of WebTics is to support SMEs (Small to Medium Enterprises), which usually do not need the accuracy of this kind of synchronization. To assist in synchronizing different events the logging system allows the developer to register a pass phrase with the logging system after opening a connection. The secondary computer system sends the logging system the same pass phrase and is given the Session and Game IDs to complete the pairing. The second system zeros its internal timer and then sends all telemetry events with the Session, Game and local time stamp indicating the time since pairing.

The metric server inserts all messages into the database in order, with a time of arrival timestamp. This, along with the IDs and a parameter including local timestamp information, will provide enough data to enable analysis. Machines used in this style of system will not typically be 'in the wild', they will be in-house research machines within companies or research labs. In this situation the computers are probably on the same LAN, minimizing network delays. With a dedicated web server to handle all log events promptly, this lightweight, easily established synchronization will be acceptable for most game event analysis.

10.7 Conclusion

In this chapter we have discussed the design of an open source telemetry and metrics system, called WebTics. This system is designed to be lightweight and easily integrated into a game development project. The open source nature of the project will result in significant additions to the tools described here. The description of the design decisions and the principles behind the development of WebTics will hopefully help the reader understand the reason for specific features. The core principle is to provide tools to the developer using simple open formats.

We are currently using WebTics as part of our game development research projects. As such there are plans for several additions to the system, including javascript and linux from ends, tutorials for using the data with Knime and Weka, and potentially an option for the backend database to allow for NoSQL, which would allow for much larger data sets.

There are already a large number of game metrics tools available, but we intend this tool to fill the role of an open source project for small companies to start to use telemetry and metrics. Once the needs are more than what WebTics can provide, it is probably time to pay for professional support, and use a company to provide your metrics tools. For the latest versions of WebTics clone the project from the GitHub repository git://github.com/SimonMcCallum/WebTics.git or fork the project and contribute to the on going development of WebTics.

About the Authors

Dr. Simon McCallum: The background for this middleware tool comes from a combination of academic interest and corporate necessity. Having started researching games for health as an academic at the University of Otago in 2003, it became clear that recording player actions for review and evaluation was essential for any scientific validation of the projects we were conducting. In 2007, I decided to live in the trenches of game development, get a commercial job in Norway. One of the companies I worked for was a game distributor, with an in-house game development team. In this team one of the projects we worked on was the design of a middleware metrics tool to provide to small developers as a service attached to our distribution agreement. This was to be used internally as well as supporting other developers. 2008 was not kind to game companies. Thus, I returned to academia and started teaching game development. The importance of game metrics stayed with me and I used it in student assignments to teach database at undergraduate, and game technology at Masters level. This API is a result of this combination of researcher need, commercial imperative, and educational tool.

Jayson Mackie. With experience over the last 15 years working with Governmental databases, high-uptime telecommunication software and data driven games, I have been in many situations where a simple API to enable customer or player machines to log extensive data to a central site would have aided performance tuning, debugging or balancing. I have also been involved in research projects where logging and tracking individual test installations was required. This lightweight API provides a very useful set of logging functions without placing major demands on the software developer.

Appendix: WebTics API

In this appendix the technology and Application Programming Interface (API) is documented in more detail. The documentation is intended to be useful for developers who have decided to consider using the system, or wish to see how we have implemented our system to improve their own metrics systems.

Selecting Technology

The API discussed in this chapter is a minimal event and message-passing interface using HyperText Transfer Protocol (HTTP) for all messages. A small library implemented in C++ provides the client-side functionality and a back-end for storing the data is implemented using PHP/MySQL and an Apache server (WAMP/LAMP). Using HTTP for system communication allows easier cross platform integration

and development. The only assumption is that there exists a network connection and network handling within the operating system.

Advantages:

- A simple API provides most of the functionality needed in most projects.
- It does not require the user to write any network routines.
- Text-based message-passing makes debugging easier.
- The front-end client can be developed in any language with networking.

Disadvantages:

- This system does not maintain a permanent direct connection, therefore the messages and replies may not arrive or may arrive out of order. They are time-stamped and can be reordered after-the-fact.
- Requires some knowledge of SQL and PHP for editing the visualization and data mining system.
- Not GUI-based, so harder for designers to use.

How to Start

1. Create a local Database and web server for testing. Download the latest version of a LAMP/WAMP server. (The Uniform Server is a good one currently)
2. Download the metrics library from GitHub. This will give you the latest stable version of the API.
3. Run the SQL build script included in the metrics API download either locally or on a remote server. (If you want to make changes to the default logging system you can edit the setup file).
4. Place the php and html files either in your local root or on the server you are using.
5. Include the header files and the library files in your project. Make sure you follow the instructions for setting up the metrics class, in the README at the top level of the project.
6. Start logging events from your game using the LogEvent calls.
7. View your data using phpmyAdmin or the realtime visualization tools in the visualization path of the metrics site included in the API.

Overview of the API

Using this system requires the inclusion of one C++ header, GameMetrics.h. The system uses a second header file, GameMetricsDefaults.h, containing the enumerated types and labels describing the events to be logged. This header is not included by the programmer, but is loaded as part of the GameMetrics.h file. It may need to be included elsewhere to provide the event types to other

program files. The functionality is provided by adding a single source file, GameMetrics.cpp. Alternatively the static library GameMetrics.lib may be linked into the project.

What follows is a summary of the usage of the API we discuss. This process is relatively simple as you only need to include a couple of setup methods for identification and tracking of the player and game session. The Game Metrics class has been implemented using the Singleton design pattern. This enables the metrics systems to be accessed from any location in the program, and controls the problem of multiple instantiations.

Creation of the metric system does not establish a connection to the logging system. A session must be opened explicitly. Any LogEvent() calls used to log an event before the session is opened will have no effect.

Session Order

We will explain each part of the API based on how you would use the system. The pseudo-code for the general structure of a program setting up a logging session is:

1. Get the instance of the GameMetrics singleton
2. Initialize the metric system
3. [optional] Check Authorization if required with unique ID
4. [optional] Request and set authorization if not already given for unique ID
5. Open metric session with unique ID
6. [optional] Register version number and event types if required
7. [optional] Get game parameters (type 1)
8. [optional] Start play session
9. [optional] Get game parameters (type 2)
10. While (the game is playing)
    …
    Log events
    …
11. [optional] Stop play session
12. Close metric session

Get the Instance of the Singleton

This method must be called to create the GameMetrics system. The system has been implemented as a Singleton so there is no public constructor. After creation further calls to GetInstance() will return a pointer to the GameMetrics system that has already been created. The calls to GetInstance() may be made from any location in the program. If the system has not previously been initialized, then passing autoInitialise as true will instruct the system to use the default server URL and PHP path defined in GameMetricsDefaults.h. If false is used the developer must use the Initialise() method. The default value is true.

```
  GameMetrics* metricsSystem = GameMetrics.GetInstance
(autoInitialise);
```

The addition API calls will be prefixed with metricsSystem to reflect their usage.

Initialize the Metric System

If the system has not been auto-initialized, or for any reason needs to be reinitialized this method may be used. The sever URL and the PHP path are passed to the system.

```
  metricsSystem->Initialise(serverURL, serverPHPPath);
  authorised = metricsSystem->IsAuthorised(uniqueID);
```

Request and Set Logging Authorization

After the programmer has obtained permission from the user via any mechanism available to them this method is used to update the database. The unique ID for this application is stored with the version number of the application that has been given permission. The version ID string may be empty if authorization per update is not required.

```
  metricsSystem->SetAuthorised(uniqueID,trueFalse,
versioned);
```

Open a Metric Session

This method is called near the start of the program. This sets the timer in the Metric system to zero. All messages sent to the logging system for this session will be time stamped with an offset from this point. At this stage you have to decide whether to track the individual user from session to session. For maximum anonymity, either required by law in a given location or because the user-base will be reluctant to participate, the logging system can be run with a session key linked only to the specific game session. This key allows events within the session to be linked but has no information linked to the user. Over a series of collected data events you will still be able to analyze how the player population behaves but will not be able to trace changes in a specific series of play sessions over time.

The alternative is to send some identifying information to the metric server to be linked to the session ID. There are many options for this. A registered game could sent its serial number for example. A player may be logged into a game environment such as Steam or your own game servers. Alternatively the user may be playing the game within Facebook or a similar social networking system. The availability of identifying information to your metrics system, and your obligation regarding storage and recording will be bound by the terms of service of the overall environment.

The benefit of being able to identify a specific game installation is the tracking of a player over time. The play actions of a group of players with a high win ratio, or

high scores allows you to identify strategies that permit the player to beat the game, and guide where the rule may need to be tweaked for balance or to provide a continual challenge. The metric session ID and play session ID will not be consistently associated with a single player so an installation ID is required.

Further discussion on using the API will assume some persistent identification of a game installation between game sessions.

```
metricsSystem->OpenMetricSession(uniqueID);
```

Register Version Number and Event Types if Required

This method sends all the current event labels to the database with the current version number. There are two forms, one without parameters, which sends the values defined in GameMetricsDefaults.h and the other which passes two other user defined arrays of strings to the database.

```
metricsSystem->RegisterEvents();
metricsSystem->RegisterEvents(versionID,
        eventsArray, numberOfEvents,
        subeventsArray, numberOfSubevents);
```

Getting the Game Parameters (1)

This is an optional action. If you chose to write your game to have parameters that can be reinitialized during game play you may return a new set of parameters to the game. These parameters may control any aspect of the game, from the points scored for each action to decision thresholds for the AI system or new build cost/times to alter the game balance. A game can receive one of a predefined set of parameters, which is recorded in the metric system. The outcome is observed or the user is questioned about how they enjoyed the play experience. This is of use especially to game designers or developers of serious games.

The text parameter is there only as an option, it may be left NULL. It allows a the current game installation or runtime information to be sent to a more complex back-end system. This information can be used by the back-end system to evaluate which set of parameters should be returned to the game. It is the responsibility of the programmer to manage the memory of this string and delete it after the string has been used.

```
  string* parameters = metricsSystem->RequestParameters
(dataString);
  string* parameters = metricsSystem->RequestParameters();
```

Start a Play Session

If a game has sets, levels or other well defined breaks in timing or difficulty it is useful to have explicit markers between these stages. Starting a play session places a marker in the metric system that may be used as a delimiter when the metrics are later analyzed.

If the game is a continuous experience then this marker will add no additional clarity to the metrics collected. You may choose to issue StartPlaySession() immediately after OpenMetricSession() and StopPlaySession() immediately before a CloseMetricSession() for completeness, or ignore the Start/Stop Play Session events.

```
metricsSystem->StartPlaySession();
```

Getting the Game Parameters (2)

This is a repeat of the game parameter request that can be performed at program start up. It is again optional. Parameters could be updated on a per set/level or chapter basis by requesting updates at the beginning of these sessions, while requests sent as start up would be used for the entire game session. It is possible to use the call multiple times, with major game parameters established for the entire game session at start up, and minor tweaks on either a random basis or determined by player behavior at the start of each chapter of the game.

```
string* parameters = metricsSystem->RequestParameters
(dataString);
string* parameters = metricsSystem->RequestParameters();
```

Logging Game Events

The metrics system uses a single call to log events, where the developer includes the game event data, and the system adds the logging and game IDs, the logging time and sending time timestamps.

The full event method takes seven parameters to offer flexibility in how it is used.

```
metricsSystem->LogEvent(type, subtype,
    xInt, yInt, zInt,
    valueDouble, dataString);
```

There is also a debugging variant of this call. The goal of the debugging version is to support logging during development. The debug version will only send messages when the game is compiled with the _DEBUG flag set in the compiler. The variant allows the developer to put event logging in the program for functional testing which can be compiled out on release.[5]

```
metricsSystem->LogEventDebug( type, subtype,
    xInt, yInt, zInt,
    valueDouble, dataString);
```

The developer may implement wrapper methods for any required subset of the parameters which uses the method above with placeholder values to indicate a field is

[5] This is implemented as an immediate return from the `LogEvent()` method in the library without performing a send. There is a small computational cost to this method. If the developer wishes to have a non-debug version completely removed `LogEvent()` method they will need to use compile time code exclusion with `#ifdef ... #endif` compiler directives.

unused. A selection of wrapper methods are already provided by the API. The wrapper methods are also available for LogEventDebug() versions. The default value for unused integers and doubles is –999999, the default value for an unused string is "".

```
metricsSystem->LogEvent(type, subtype,
       xInt, yInt, zInt,
       valueDouble, dataString);
metricsSystem->LogEvent(type, subtype, xInt, yInt, zInt);
metricsSystem->LogEvent(type, subtype, valueDouble);
metricsSystem->LogEvent(type, xInt, yInt, zInt);
metricsSystem->LogEvent(type, valueDouble);
```

Type and subtype are developer defined. They are logged in the database and it is left to the developer or designer to define the meaning of the stored values. For clarity in the source code it can be advantageous to use enumerated types for the event type and subtype, e.g.:

The player was killed by a grenade at location 12,20,30 and they sustained 70 points of damage;

```
metricsSystem->LogEvent( EventTypes::PlayerDeath,
                         EventSubtypes::Grenade,
                           12, 20, 30, 70.0);
```

The player picked up 10 grenades at location 12,20,30;

```
metricsSystem->LogEvent(EventTypes::PlayerWeaponPickup,
                          EventSubtypes::Grenade,
                               12, 20, 30, 10.0);
```

The player respawned at a location;

```
metricsSystem->LogEvent(EventTypes::PlayerRespawn,
12, 20, 30);
```

The player fell 42 m;

```
metricsSystem->LogEvent(EventTypes::PlayerFall, 42.0);
```

Stop Play Session

This should be used to place a marker in the metric system to terminate a defined period of gameplay. After issuing this command the play session is no longer a valid. Closing a play session twice has not additional effect.

```
metricsSystem->StopPlaySession();
```

Close Metric Session

This method is used to place a marker in the metric system to indicate the game session has terminated normally. The absence of a close event before the presence

of a new open event from the same user may indicate an unscheduled program exit such as a forced quit or program crash. Closing a session twice has no additional effect.

```
metricsSystem->CloseMetricSession();
```

Data Storage

The data collected by the logging system is primarily in a single table in your testing database. The table is set up to have indexing on event-type and sub-type which will aid the most common queries. Given that this API is for small developers and academic research, the speed of modern hardware is quite capable of accessing the volume of data normally collected. If there are problems with the speed of the database, then there would need to be time spent improving the back-end implementation of this system.

The data has been logged with several hierarchical data types which allows rapid and accurate partitioning of the views of the data-set. Firstly the logging sessions have been initialized with a unique ID, and each play session's start and end has also been logged. Within the play sessions, event-type and event-sub-type, which are developer defined and arbitrarily extensible, provide as many game event categories as required.

The system expects to have a network connection but a failure to find a network can be accommodated. If a network is not available, or the OpenSession() attempt fails the system can write log events to a file on disc. On a successful connection to the metric server, if a log file exists on disc it is opened and sent to the server with a marker to indicate it was an offline session. The timing of events relative to the start of the logging session is maintained as is the order so the session can still be used for data mining. Finally all events are logged with a time stamp to allow small parts of a game session, e.g. the 30 s prior to a repeatable crash event, or the first 60 s of an emerging dominant strategy to be viewed.

For viewing the data we recommend that repeated queries to the database should be defined as a database view for efficiency. The data is also well suited for access from a web-based Javascript form. This query may also be built up directly in GUI tools such as phpSqlAdmin.

References

Altman, D. G., & Bland, J. M. (1995). Statistics notes: Absence of evidence is not evidence of absence. *British Medical Journal, 311*(7003), 311–485.

Askedal, Ø. (2011). *Are elderly sufferers of dementia able to play a reminiscence game on a tablet device independently?* Gjøvik: Gjøvik University College.

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., et al. (2008). KNIME: The Konstanz information miner. In *Data analysis, machine learning and applications* (pp. 319–326).

Bostock, M., Ogievetsky, V., & Heer, J. (2011). D^3 data-driven documents. *IEEE Transactions on Visualization and Computer Graphics, 17*(12), 2301–2309.

Charmaz, K. (2003). Grounded theory. In N. K. Denzin & Y. S. Lincoln (Eds.), *Strategies of qualitative inquiry* (2nd ed., pp. 249–291). Thousand Oaks: Sage Publications.

Derosa, P. (2007). Tracking player feedback to improve game design. *Gamasutra* http://www.gamasutra.com/view/feature/129969/tracking_player_feedback_to_.php. 7 Aug.

Drachen, A., Canossa, A., & Yannakakis, G. N. (2009). *Player modeling using self-organization in tomb raider: underworld.* Paper presented at the proceedings of the 5th international conference on Computational Intelligence and Games, Milano, Italy.

Dustin, E. (2002). *Effective software testing: 50 specific ways to improve your testing*. Boston: Addison-Wesley Professional.

Fields, T., & Cotton, B. (2011). *Social game design: Monetization methods and mechanics*. San Francisco: Morgan Kaufmann.

Gee, J. P. (2003). *What video games have to teach us about learning and literacy*. New York: Palgrave MacMillan.

Göbel, S., Hardy, S., Wendel, V., Mehm, F., & Steinmetz, R. (2010). *Serious games for health: Personalized exergames.* Paper presented at the proceedings of the international conference on Multimedia, Firenze, Italy.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter, 11*(1), 10–18.

Henderson, C. (2008). Managing software defects: defect analysis and traceability. *SIGSOFT Software Engineering Notes, 33*(4), 1–3.

Isbister, K., & Schaffer, N. (2008). *Game usability: Advancing the player experience*. San Francisco: Morgan Kaufman.

Johnston, C. (2012). Diablo 3: Give a knight a shield, and he kicks you from the game. *Ars Technica.* http://arstechnica.com/gaming/2012/05/diablo-3-give-a-knight-a-shield-and-he-kicks-you-from-the-game/

Kim, J. H., Gunn, D. V., Schuh, E., Phillips, B., Pagulayan, R. J., & Wixon, D. (2008). *Tracking real-time user experience (TRUE): A comprehensive instrumentation solution for complex systems.* Paper presented at the proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, Florence, Italy.

Lewis, J. M., Trinh, P., & Kirsh, D. (2011). A corpus analysis of strategy video game play in starcraft: Brood war. In L. Carlson, C. Hölscher, & T. Shipley (Eds.), *Proceedings of the 33rd annual conference of the cognitive science society* (pp. 687–692).

Mandryk, R. (2008). Physiological measures for game evaluation. In K. Isbister & N. Schaffer (Eds.), *Game usability: Advice from the experts for advancing the player experience* (pp. 207–235). San Francisco: Morgan Kaufmann.

Pagulayan, R. J., Keeker, K., Wixon, D., Romero, R. L., & Fuller, T. (2003). User-centered design in games. In J. A. Jacko & A. Sears (Eds.), *The human-computer interaction handbook: Fundamentals, evolving technologies and emerging applications* (pp. 883–906). Mahwah: Lawrence Erlbaum Associates.

Rabin, S. (2000). The magic of data-driven design. In M. DeLoura (Ed.), *Game programming gems*. Boston: Charles River Media.

Sawyer, B., & Smith, P. (2008). *Serious games taxonomy.* Paper presented at the Game Developers Conference, San Francisco.

Sieber, J., McCallum, S., & Wyvill, G. (2009). BallBouncer: Interactive games for theater audiences. In *Chi '09 workshop – Crowd-Computer interaction* (pp. 29–31). Boston.

Thompson, C. (2007). Halo 3: How Microsoft labs invented a new science of play. *WIRED MAGAZINE* (15.09).

Yannakakis, G. (2012). Game AI revisited. In *Proceedings of ACM computing frontiers conference*.

Zoeller, G. (2011). MMO rapid content iteration. In *Game developer conference,* San Francisco.

More Resources

- The programming principles used to develop this API are a mix of our own experience from both industry and academia. In addition to the references above we recommend additional reading.
- The code in this chapter uses the singleton design pattern. The seminal book for design patterns is *Design Patterns: Elements of Reusable Object-Oriented Software*, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, ISBN 0-201-63361-2.
- For an introduction to PHP/MySQL and Database we recommend *Learning PHP, MySQL, and JavaScript*, Robin Nixon, ISBN: 0-596-15713-4.
- Those with some experience may find it useful to read *PHP5 and MySQL Bible*, Steve Suehring, Tim Converse, and Joyce Park, ISBN: 0470384506.
- Purely as a language reference http://www.w3schools.com/ provides information for all the web related programming languages used in this implementation.
- For research on game usability and testing the work of the Microsoft Studios Research is excellent for both collecting and analyzing game experience.
- The work of Anders Drachen, Alessandro Canossa, and Georgios Yannakakis at the ITU in Denmark is an excellent starting point for work in the research community on game metrics. Particularly the recent work on Game Metrics Mining with IO Interactive. http://game.itu.dk/index.php/Game_Metrics_Mining