Exploring Game Analytics Solutions for Data-Driven User Research in Indie Studios

by

Thomas Galati

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in

Computer Science

The Faculty of Science
University of Ontario Institute of Technology

Supervisor:
Dr. Pejman Mirza-Babaei

August 2017

© Thomas Galati, 2017
Abstract

User research methodologies can be applied to games for the purposes of evaluating and improving them. Performing user testing during game development can highlight issues in gameplay early on, generally improving the end product. However, this evaluation can be costly, requiring frequent user tests and analysis. Because of this, small-size studios can find it difficult to perform formative evaluations of their games, simply due to a lack of resources. In this thesis, analytics methodologies in a mixed-methodology approach are proposed to help optimize the user research process, helping developers make informed choices to improve their games. This thesis features a framework which can be used to help assist these small-size developers in choosing from different solutions. Using this framework, three solutions are discussed and recommendations are given based on developer needs.

Keywords: Games User Research, Mixed-methods, Analytics, Data-Driven User Research
Acknowledgments

There are many I would like to thank, without all of whom this thesis would not be completed today. First and foremost, my supervisor, Pejman Mirza-Babaei, who tirelessly supported me in my research and believed in me. I would like to thank the Faculty of Science and the Faculty of Business and Information Technology at UOIT. I would not have completed this thesis without my professors, especially Andrew Hogue and Bill Kapralos, for their support of the Games and Entertainment Media Research (GAMER) Lab, and all the research conducted within it. I would also like to thank the members of the UOIT GAMER lab, including, Wallace Balanuic, Cameron Chodan, Brandon Drenikow, Nathan Gale, Mike Gharbharan, Naeem Moosajee, James Robb, Rob Shewaga, and Alvaro Uribe, who I call not only my colleagues, but my friends. Also at UOIT, I give special thanks to Brandon Le, Erix Lemieux, Bishop Osbourne, and Ben Whittaker, for their assistance in developing YoGURT.

I am very grateful for my time spent at Execution Labs in Montréal, and so I would like to thank the team, Behrouz Bayat, Lysiance Charest, Jason Della Rocca, Derek Elliot, Keith Katz, and Astrid Rosemarin, for giving me the opportunity of a lifetime.

Finally, I would like to thank my friends and family, for being by my side as I completed this thesis. I would especially like to thank my mom and dad, who supported me throughout not only the time I was working on this thesis, but through my whole life. I would not be where I am today without them.
List of Publications

2017:

# Contents

1 Introduction .................................................. 2  
1.1 Overview .................................................. 2  
1.1.1 Why Focus on Video Games? .......................... 3  
1.1.2 Games User Research ................................. 6  
1.1.3 Independent Studios and Developers ............... 8  
1.2 What Problem is this Thesis Solving? .................. 9  
1.3 Outline of Thesis ....................................... 10  
1.4 Summary (Chapter 1) .................................. 11  

2 Related Works ............................................... 13  
2.1 Outline .................................................. 13  
2.2 Video Games and the Industry ......................... 13  
2.3 Games User Research .................................. 14  
2.3.1 Understanding Third-Wave HCI and GUR .......... 15  
2.3.2 Benefits of User Testing ........................... 16  
2.3.3 Methods in GUR and HCI ........................... 17  
2.4 Analytics ............................................... 19  
2.4.1 Telemetry in Games ................................. 19  
2.4.2 Example of Analytics use ........................... 20  
2.4.3 Visualizing Data .................................... 21  
2.4.4 Contextual Telemetry ............................... 22  
2.4.5 Tools to Assist Analysis ............................ 22  
2.5 Mixed-Method User Research ........................... 23  
2.5.1 Analytics in Mixed-Method User Research ....... 24  
2.6 Considerations for Analytics Tools .................. 25  
2.6.1 A Comparison Framework for Analytic Tools .... 26  
2.7 Summary (Chapter 2) .................................. 29  

3 Exploring Analytics Solutions .............................. 30  
3.1 Overview ............................................... 30  
3.2 Introduction ............................................ 30  
3.3 Motivation .............................................. 31  
3.4 Developing YoGURT ........................................ 33  
3.4.1 YoGURT Plugin ..................................... 33  
3.4.2 YoGURT Web Server ................................ 36
3.5 Using YoGURT ........................................... 38
  3.5.1 Usage Scenario ..................................... 41
3.6 Discussion ............................................. 42
3.7 Summary (Chapter 3) .................................... 43

4 In-House Solution ......................................... 44
  4.1 Overview ............................................... 44
  4.2 Introduction ............................................ 44
  4.3 Motivation .............................................. 45
    4.3.1 Studio Needs ...................................... 45
    4.3.2 Stream Game ...................................... 46
    4.3.3 Analytics Pipeline ................................. 47
    4.3.4 Automated Reports ................................ 47
    4.3.5 Requirements ..................................... 48
  4.4 Development ........................................... 51
    4.4.1 Widgets .......................................... 51
    4.4.2 Data Sources ..................................... 53
    4.4.3 Visualization ..................................... 54
  4.5 Case Study ............................................. 56
    4.5.1 User Test ......................................... 56
    4.5.2 Dashboard Setup .................................. 57
    4.5.3 Automated Report ................................ 59
    4.5.4 Using Autodash ................................... 60
  4.6 Discussion ............................................. 61
    4.6.1 Comparison Framework ............................. 62
  4.7 Summary (Chapter 4) .................................. 64

5 Third-Party Solution ...................................... 65
  5.1 Overview ............................................... 65
  5.2 Introduction .......................................... 65
  5.3 Motivation .............................................. 66
    5.3.1 Studio Needs ...................................... 67
    5.3.2 Requirements ..................................... 68
    5.3.3 DeltaDNA ......................................... 69
    5.3.4 Stream Game ..................................... 70
  5.4 Usage .................................................. 71
    5.4.1 Case Study using DeltaDNA ....................... 71
    5.4.2 Analysis .......................................... 72
  5.5 Discussion ............................................. 75
    5.5.1 Comparison Framework ............................. 75
  5.6 Summary (Chapter 5) .................................. 77
6 Discussion & Conclusion

6.1 Overview ............................................... 78
6.2 Discussion ............................................. 79
   6.2.1 Affordability ........................................ 79
   6.2.2 Shareability ......................................... 81
   6.2.3 Data Refresh Rate .................................... 82
   6.2.4 Custom Data Analysis ................................. 82
   6.2.5 Visualization Quality .................................. 83
   6.2.6 Customizability ....................................... 84
6.3 Thesis Contributions ....................................... 85
   6.3.1 Comparison Framework ................................. 86
   6.3.2 Overall Comparison and Recommendations .............. 86
   6.3.3 In-House VS Third-Party ............................. 87
6.4 Limitations and Future Work ................................ 89
   6.4.1 Small-Size Studio Representation ...................... 89
   6.4.2 Creating an Improved Solution ......................... 89
   6.4.3 Other Applications .................................... 91
6.5 Conclusion ............................................... 91

References ................................................. 93
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Rough YoGURT database diagram</td>
<td>37</td>
</tr>
<tr>
<td>3.2</td>
<td>YoGURT game list</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>YoGURT instances list</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>YoGURT session list</td>
<td>40</td>
</tr>
<tr>
<td>3.5</td>
<td>YoGURT event list</td>
<td>40</td>
</tr>
<tr>
<td>3.6</td>
<td>YoGURT event count graph</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>Autodash widget showing time</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Autodash widget showing a Google Chart</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>Autodash widget showing HTML table</td>
<td>55</td>
</tr>
<tr>
<td>4.4</td>
<td>Autodash use case, interactions over time</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>Autodash use case, interactions over type</td>
<td>58</td>
</tr>
<tr>
<td>4.6</td>
<td>Autodash use case, currency over time</td>
<td>59</td>
</tr>
<tr>
<td>4.7</td>
<td>Autodash sample</td>
<td>60</td>
</tr>
<tr>
<td>5.1</td>
<td>Currency Spent per Item (Phase 1)</td>
<td>73</td>
</tr>
<tr>
<td>5.2</td>
<td>Currency Spent per Item (Phase 2)</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Currency Spent / Received by Spectators</td>
<td>74</td>
</tr>
</tbody>
</table>
## List of Tables

3.1 Common Parameters in Analytics Solutions .......................... 34  
4.1 Comparison of dashboarding tools .................................... 50  
4.2 Properties of the Autodash widgets ................................. 52  
6.1 Overall comparison of different approaches ....................... 87  
6.2 Recommendations based on tools ................................... 88
Chapter 1

Introduction

1.1 Overview

This thesis presents and discusses the benefits of the application of analytic methods in Games User Research (GUR), with a specific focus on accessibility and affordability for independent or small-team game developers. GUR is a field of research dedicated to evaluating games based on user research paradigms. However, GUR is not evenly accessible by all developers, an issue which is largely present because independent game developers do not have the same resources as a AAA game studio, and cannot afford to perform the same level of GUR as the larger studios. This issue is compounded by a general lack of specialized user researchers trained in the field. Large companies typically can afford to have a user research team on staff, as they may be publishing several games at any given time, all of which need user testing performed. Smaller studios are much more focused on a single product or deliverable, and it is therefore much more unlikely for them to have a dedicated team of specialists performing GUR at all stages of development [1]. This leaves an indie team at a distinct disadvantage, as they are often unable to perform adequate user testing, stemming from a lack of knowledge and specialists.
Understanding the above disadvantage of independent teams, this thesis acts as a reference guide for these smaller size studios, so that they may approach having a usable GUR framework, given their resources. GUR methods often require lots of time and other resources. Thus, the focus of this thesis will be largely applying game telemetry and analytics methods in mixed-method approaches to optimize the GUR process. There is a discussion presented of different tools that can be used to achieve this goal, ranging from in-house solutions to ones that may be available for a paid fee. The benefits of each of these methods will be discussed, and the strengths and weaknesses highlighted. The use of these tools to aid in telemetry gathering and analysis can optimize and streamline the GUR process for the smaller gaming studios, and demonstrate more affordable and accessible GUR methods. This thesis does not aim to suggest that the commonly used GUR methods are subpar, or that they need replacement, but to instead simply offer affordable and accessible solutions for small, independent studios who do not have the resources of a larger company.

1.1.1 Why Focus on Video Games?

Video games, virtual simulations, and interactive experiences make up a gigantic industry. According to a report on the game industry published in 2017 [2], video game software sales exceeded $24.5 billion in 2016. In the United States, there are almost 2,500 video game companies, providing over 220,000 jobs. The gaming industry added $11.7 billion to the U.S. GDP in 2015, (or roughly $36 per capita, calculated from a 2015 population estimation from the U.S. Census Bureau [3]). Based on a report provided by the Entertainment Software Association of Canada (ESAC) in 2015 [4], the gaming industry adds $3.0 billion to the Canadian GDP each year (or roughly $85 per capita, according to the 2016 census provided by Statistics Canada [5]). As of 2016, there are 472 active studios in Canada directly employing 20,400 people [6].
There is value, therefore, in pursuing this field simply due to the value added to the Canadian economy.

Even if one were to not consider the economic impact of the gaming industry, there are still a number of factors that make this topic such a popular one for the purposes of research. The highlighted factors below briefly suggest reasons why gaming is so large, and why it is beneficial to research methods to improve games:

**Demographic:** Video games have recently attracted audiences that traditionally have not been gamers. This can be seen especially with the rise of female gamers, who make up 49% of Canadian gamers [6]. The average Canadian gamer is 36 years old, and plays games for 11 hours per week [6]. As the number of players continues to grow it becomes more imperative that their gaming experiences are positive to ensure continued growth, demonstrating a need for GUR to ensure good quality experiences.

**Accessibility of Developers:** Just as it is easier for people playing games to get started in gaming, it is easier for developers of games to start making games. The rise of efficient, free to use game engines such as Unity and Unreal [7, 8] have made the creation of games and other digital experiences much easier. There is a wealth of resources available regarding both of these engines, so developers with limited industry experience can find themselves able to create a game. These free to use tools make it easier for independent teams to start making games. Due to the increased access of development, there are bound to be more developers, and therefore more games. It is important for indie games to deliver a good quality game and positive player experience to survive in the competitive environment [9]. Hence, there is a need to adopt GUR processes and methods specifically for smaller teams and their respective budgets and time frames.

**Platforms:** One need look no further than the mobile phone in one’s own pocket to see how easy it is for someone to start playing games. Almost everyone has access
to a platform which can be used to play a incredibly large selection of games, with 89% percent of Canadian adults owning a mobile device, and 90% of Canadian adults owning a personal computer (PC) [6]. In fact, 72% of Canadians reported playing on their mobile device within a 4 week period, up from 20% in 2012, and 65% reported playing a game on their PC within the same 4 week period, up from 58% in 2012 [6]. Compare this to the past, where it would take a significant investment to gain access to a game console and a game to play in the comfort of one’s own home. Even standard PCs can play many available games thanks to platform-independent game engines like Unity [7], which can develop for web players. As platforms for gaming are more accessible, there are more potential players for games. To retain these potential gamers, game quality must be assured, demonstrating a need for GUR methods.

Cost: Free-to-Play (F2P) monetization strategies have further reduced the initial cost barrier to start playing a game. It is no longer required to spend a large amount of money up front in order to enjoy some of these games, making it easier for people to start playing F2P games. The lack of requiring any form of purchasing makes these games very attractive, and invites more and more people to try out games. Similarly to how platforms for gaming are more accessible, F2P monetization allows for games to reach a wider audience as well. Many of these F2P games are focused on the First-Time User Experience (FTUE), or first hour of play specifically [10]. This is to ensure that the first time a player plays a game, it is an enjoyable experience, which keeps them playing.

Distribution: To build off of the points stated above, it is easier for game developers to distribute games, as well as it is for players to play them. This is due to the use of digital distribution; there is no longer a need for a physical copy of a game to be made. Thanks to the power of the internet and online services like Steam¹,

¹store.steampowered.com/
Origin\textsuperscript{2}, and uPlay\textsuperscript{3}, potential players of a game can simply download and install a game instead of having to go to a brick-and-mortar location to purchase a copy. This severely reduces costs for developers, as they now do not necessarily need to go through a typical publisher in order to get their games to the masses. In addition to this, there is no material overhead to selling a game through digital channels. Furthermore, online services like Google Play\textsuperscript{4} and the Apple App Store\textsuperscript{5} continue to make it easier than it ever has been for developers to publish their games.

After having mentioned all the positives of the gaming industry, which make it very accessible and easy to start developing and playing games, it would be remiss to not mention that designing and developing video games is a costly, demanding process. There is no designated streamlined process for game development that every single company follows. This is because a streamlined pipeline would give a competitive advantage to the company that has developed it, and this would simply not be public knowledge.

Understanding the expanding gaming market and the importance it has on the economy, it can be seen that ensuring the quality of games is important. This thesis discusses methods available to small-scale and independent developers to ensure the quality of their games, by demonstrating mixed-method or combined GUR approaches.

1.1.2 Games User Research

In today’s era of free to play gaming, monetization strategies, and a focus on user retention in an extremely competitive market, there is more need than ever for games to have a great user experience. To simplify this: users with a positive experience

\footnotesize{\textsuperscript{2}origin.com}  \textsuperscript{3}uplay.ubi.com}  \textsuperscript{4}play.google.com}  \textsuperscript{5}itunes.apple.com}
with a game will continue to play that game, whereas users with a negative experience will stop playing.

To gauge these user experiences, one can borrow user research methods used in the field of Human-Computer Interaction (HCI). These methods have been used to understand the usability of productivity applications as well as websites. Common methods include questionnaires, interviews, and focus groups [11]. These methods, while effective for scenarios with productivity applications, are not always as effective in game-related projects, as these methods are largely focused on usability, whereas games are more focused on the user experience. User experience is, by nature, difficult to quantify, proposing an interesting challenge for those who wish to perform a formative evaluation of games.

Games User Research (GUR) is an emerging field building upon the evaluation methods of HCI and psychology. Through conducting usability testing from HCI and user experience (UX) evaluation, GUR aims to improve gameplay experiences by bringing the game closer to designers intention [12]. As stated earlier, it is difficult to quantify user experience, so GUR methods often need multiple iterations of testing in order to produce actionable results. The goal of a GUR test is to provide a report that satisfies these conditions: (1) the playtest and information presented in the report is representative of the game demographic; (2) the report provides accurate information; (3) the report is specific in highlighting individual issues instead of general ones; (4) the report is timely, with respect to the project timeline; (5) the user test is cost-effective, that is to say that the value gained from performing GUR exceeds the cost of performing it; (6) the test must provide actionable results; and finally (7) the report is motivational to the developers [13].

Different GUR methods have different benefits, and satisfy the aforementioned conditions to varying strengths. These methods will be further discussed in later chapters of this thesis, but consider this example: observation generally provides
very complete information, making it accurate, specific, actionable, and motivational. However, it is not as timely, generally focuses on a smaller group of people so it is less representative, and is as a result potentially not as cost-effective as some other methods. Telemetry analysis, or the analysis of in-game event data, by comparison is very timely and therefore cost effective, and can scale much more easily for multiple playtesters, making it potentially more representative [14]. Mixed-method approaches, or a combination of GUR methods, can help to limit weaknesses while maintaining the strengths of the methods [15].

1.1.3 Independent Studios and Developers

According to the Entertainment Software Association report in 2017 [2], the number of digital entertainment companies reporting employment is on an upward trend from 2013, and the average number of employees per location is on a downward trend from 2013. This means that generally speaking, the average game development firm is going to have less employees than has been in the past. This suggests a trend towards more small-size or independent studios. These independent studios typically do not have the access to the large amount of resources that a standard AAA or AA company will have.

What is indie?: “Independent developer” is a term that refers to any company that does not rely on a publisher to distribute their games. Generally speaking, these are smaller studios. There are various definitions for what one might consider to be a “small-sized” developer, so it will be defined that, for the purpose of this thesis, a small-sized development studio is considered to be any studio with less than 20 employees. It is worth noting that there are studios in the 1-4 employee range, which are considered “micro-studios” [4].

Many AAA companies such as Microsoft, Sony, and Electronic Arts, have in-house playtesting teams with dedicated methodologies [16, 17]. This is effective in
these large companies, as they will often be working to publish multiple games in a development cycle, with many teams overlapping the use of a GUR team. Some companies might use third-party GUR teams like Bunnyfoot⁶, or Player Research⁷. For companies with a large enough budget to outsource the user testing, hiring dedicated specialists with knowledge of GUR processes on a short term can be quite beneficial. Unfortunately, these processes are all considered competitive advantages and are not accessible to the public. This makes it difficult for independent developers to compete, due to having a lack of specialized GUR knowledge, as well as cost of evaluation [18]. Understanding these limitations, it can be seen that there is value in creating low-cost and accessible methods of game evaluation for these independent studios [19].

1.2 What Problem is this Thesis Solving?

In today’s era of free to play gaming, monetization strategies, and a focus on user retention in an extremely competitive market, there is more need than ever for games to have a great user experience. Understanding this and the problems faced by independent studios as presented above, it can be seen that there is a need for affordable and accessible game evaluations for indies.

This thesis proposes different GUR solutions which can be used by small-size studios. As presented earlier, game telemetry analysis is timely and efficient, as tools can be written to automate the interpretation of the telemetry data, and to visualize it in a meaningful way for developers. This is beneficial, as there can be less time and resources spent on evaluation thanks to optimizations in using analytics in mixed-method user research. This thesis presents 3 such tools that have found some success being used in a variety of different situations.

⁶bunnyfoot.com
⁷playerresearch.com
This thesis contributes to the field of GUR by: (1) creating a framework to compare analytics solutions for small-size studios, (2) utilizing this framework to compare different solutions, (3) providing a discussion of the benefits of third-party and in-house solutions.

This thesis is influenced and motivated by the following research questions:

1. How can the cost of GUR be reduced for developers?

2. What analytics solutions are available to small-size studios to perform user research?

3. What recommendations can be made for small-size studios looking to perform user research?

1.3 Outline of Thesis

Chapter 2 covers related work in the field of HCI and GUR, with respect to different user testing methodologies. There is a specific focus on what can be done timely and cheaply for independent studios.

Chapter 3 discusses the creation of an in-house GUR tool, Your Games User Research Tool (YoGURT). It is a tool developed in order to address the needs of independent developers who may not have the necessary experience or knowledge in order to set up a metric analysis suite for their game. The benefit of YoGURT is that it largely automates the process of gathering telemetry. It is written as a simple C++ plugin that should be able to integrate with many existing games in-development. The goal was to create a tool that could let users have in-game events recorded within an hour.

Chapter 4 discusses the creation of another in-house GUR tool, Autodash. YoGURT, the predecessor to Autodash, while effective, had a largely lackluster visualization of data. This made analysis difficult, and time-consuming. This led to a
specific focus on automation of data representation, to optimize the process. This was performed through the use of a customizable dashboard. The dashboard calls pre-created procedures for a MySQL database containing game telemetry.

Chapter 5 discusses an adaption of DeltaDNA, a third party solution. The reason it is being mentioned is that it is the relevant competition to the ideas thus far presented. It creates and maintains a database which can be populated with game telemetry and allows for queries to be run on it. However, DeltaDNA is a paid service, and due to it being an online service, the server updating is not instantaneous. There is an amount of downtime between a user playing through a session and being able to gain access to the information from that session.

Chapter 6 presents a discussion of the different methodologies that were used, and their overall effectiveness as it pertains to creating cost-effective GUR solutions. From a combination of methodologies here, optimal solutions for game developers requiring more budgeted GUR solutions are presented.

1.4 Summary (Chapter 1)

The video game industry is becoming increasingly large. As it is continuing to become such a giant force in the world, there is value in ensuring quality in the products of the field. This quality assurance is performed through the use of GUR processes. These GUR methods typically require a degree of specialization knowledge by Games User Researchers, who know and understand how to perform the analysis. This is an effective strategy for large game companies with dedicated GUR teams. However, smaller, independent studios generally do not have the resources to have a dedicated team of GUR researchers. These smaller developers still are in need of GUR solutions to ensure the quality of their product, thereby presenting a need for a low-cost, accessible means of performing formative game evaluation. This thesis contributes a resource and guideline of available GUR methods and procedures for small-size
studios to follow, creates and utilizes a framework for comparing different analytic solutions, and presents case studies using those solutions. This is done to hopefully reduce the time and therefore cost of GUR for developers.

The next chapter provides a background of the research and justification of GUR methods, understanding player needs, and the impact of different analysis methods.
Chapter 2

Related Works

2.1 Outline

This chapter discusses related works in Games User Research, presenting literature that covers a number of insights into the fields of HCI and GUR. This is done to better frame the contributions of the thesis. This chapter starts with a discussion of the gaming industry, then follows with an explanation of Games User Research. There is a specific focus on analytics and mixed-method user research.

From this research, a framework for comparing analytic solutions is proposed. This framework is presented in the interests of small-size game developers, who may not have the resources to conduct effective user research. The proposed framework demonstrates key points of consideration for analytic tools to be used in mixed-method approaches to user research.

2.2 Video Games and the Industry

Video games and interactive experiences make up a huge industry, which is currently expanding [4]. By the end of 2014, Canadian video game companies had completed 1280 game projects, up from 910 projects in 2013, a growth of 40% in a single year. As of 2016, there are 472 active studios in Canada directly employing 20,400 people
Based on 2015 figures, 56% of Canadian game companies had between 5-100 employees, and 39% of all companies had less than 5 employees. The large amount of small-size studios is exciting for research, as it shows opportunities to develop and research solutions specifically for these studios.

However, while this industry is continuing to grow, it is not without concerns. An analysis of the top 100 iOS games shows that only the top 20 games are truly profitable, and anything outside of the top 100 will not be making enough money to support any more than a one or two developer team [20]. While this is an analysis for the Apple App store, it would not be unreasonable to assume that similar figures would be true of other distribution channels. This demonstrates a need for each game published to be of extremely high quality, in order to ensure success.

Understanding this need for high quality games, there is also a need for methods to improve user experiences or user engagement [12] to ensure the quality of these games. In larger studios, this can be accomplished by creating a user research division of the company, or an internal team of dedicated evaluation specialists, like Microsoft Labs [21]. However, with the industry trending towards smaller studio sizes [2], small-size studios are less likely to have dedicated user research specialists on their teams.

This presents an opportunity for research into evaluation methods tailored for small-sized studios. As this thesis explicitly discusses methods to optimize user research to make it more accessible and affordable for these small-size development teams, one can agree that there is value in this thesis, due to the progression of the industry.

2.3 Games User Research

Games User Research (GUR) is an application of Human Computer Interaction (HCI) research, with a specific focus on games and interactive experiences [22]. To understand GUR, first one must understand HCI. HCI is a field of research that studies the interactions of people with various digital interfaces. It is the comprehension
and application of concepts from other fields, leading to an understanding of cognition, used to design positive user interactions [23]. HCI was born from cognitive and experimental psychology, largely concerned with notions of task efficiency, work performance, ease of use, and utility [24]. The methods used in HCI for digital interfaces may also be applied to games, focusing on not “human-computer interactions”, but more specifically “player-computer interactions” [24].

2.3.1 Understanding Third-Wave HCI and GUR

A mark of the expanding field of HCI is the research into applications of technology and non-tangible factors such as feelings or experience. This pursuit of knowledge in these more abstract areas is referred to by Bødker as the “third wave” of HCI [25]. This offers a promising perspective for video games research, as both fields are concerned with the evaluation of user experiences between interactive technology and the humans using it.

In terms of current research in this field, a paper presented at CHIPlay 2014 by Carter et al demonstrates an analysis of 178 papers published at the Special Interest Group conference on Human Factors in Computer Systems (SIGCHI), with the intent of establishing the paradigms of research into HCI for games. The four paradigms which were developed over multiple workshops of iterative analysis are: operative, epistemological, ontological, and practice games research. These paradigms are presented as:

Operative games research, which is a field of research that leverages knowledge gained from the study of games or play to exert control upon the world. An example of such control is encouraging exercising or learning. This research paradigm could be simplified as the concept of gamification, or “the use of game design elements in non-game situations” [26].
Epistemological games research is focused on using games as a vehicle for understanding the use of all technologies, rather than only in the context of the unique modes of interactions or affordances of games and play, such as virtual embodiment or interfaces. To simplify, this research paradigm is categorized as studying novel ways of interacting with games (such as control schemes), and understanding player behaviour.

Ontological games research is concerned with the design and understanding of the ontology of games: rules, aesthetic, interfaces, fiction, and game design patterns. In a manner of speaking, this concerns with improving the field of games to generate play.

Practice games research that is concerned with the emergent practices and experiences that occur as a result of interaction with games or toys, or when interacting with technology with a ludory attitude. Rather than focusing on the game and its ontology (design, rules, control systems, embodiment and so forth), studies within this paradigm focus on the experience and interactions between users that occur as a result of game interactions [24].

Games user research is most closely related to practice games research, for the focus on exploring user experiences. User researchers, when attempting to evaluate user experiences, do so by designing user tests.

2.3.2 Benefits of User Testing

In order to validate the research this thesis provides, it needs to be demonstrated that user testing has a noticeable improvement on the base product, with a specific focus on games. Fortunately, Mirza-Babaei et al [22] report a statistically significant increase in perceived game enjoyment after performing and acting on user testing. The authors of the paper seek to demonstrate the usefulness of user tests (UTs) for game designers, and propose a Games User Research (GUR) method called Biometric
Storyboards (BioSt). BioSt is a data visualization tool made for the Unity game engine that links game events and a player’s physiological measures to provide context to telemetry data, with the intent of creating a better gameplay experience. The researchers create 3 game development teams of 2 designers each to modify the game MoS based on different user testing methodologies. The methodologies are: (1) No UT, or pure developer intuition, (2) Classic UT, or surveys, interviews, etc, and (3) BioSt UT, using physiological measures. The modified versions of the game are rated by a group of players using four surveys (PANAS, SAM, SUS and an in-house Likert questionnaire). It was found that the BioSt UT and Classic UT were not significantly different by the opinions of players, but are both seen as having vastly improved the game over the pure developer intuition. From this source, it can be seen that there is value in performing user testing.

2.3.3 Methods in GUR and HCI

User research methods in GUR and HCI generally seek to gain data from users. This data can be split into two different categories: *behavioural* data and *attitudinal* data [27]. Behavioural data refers to observable data based on performance, or the actions taken by a participant that one can measure. Examples of such data include the time it takes to complete a given task, the number of attempts before succeeded said task, etc. Attitudinal data refers to opinions of a participant, such as subjective ratings from questionnaires.

Another way of splitting data from users is to divide them based on whether or not they are *qualitative* data or *quantitative* data [28]. Qualitative data is “soft” data, or subjective data, typically gained through methods like interviewing. Quantitative data is “hard” data, or objective data, typically gained through methods like surveys and questionnaires, or through telemetry.
This thesis will present a selection of some methods used in HCI and GUR, and discuss whether they are behavioural or attitudinal, and whether they are qualitative or quantitative.

**Observation**

Observation is a method by which user researchers observe a user simply using the product. In gaming, this would be a player playing the game. This can be done as a live observation, with the researcher observing a participant in real time, or as a video-recorded observation. Observation generally provides behavioural data, and can be a relatively inexpensive form of user research [29]. Depending on how the observation is carried out, it can provide both qualitative (eg: user appears flustered on level 2) and quantitative (eg: user completed level 2 in 10 minutes) data.

**Questionnaires**

Questionnaires and surveys are methods by which user researchers can gain quantitative data. They are a series of questions designed to elicit specific kinds of answers from users. Some require a simple YES/NO answer, where others may ask for a longer responses or comments [23]. They can be taken on printed paper, or through digital means, like Google Forms\(^1\). Questionnaires can give either behavioural or attitudinal data. Questionnaires are a relatively simple and inexpensive means of gaining quantitative data, so they can be used to great extent by small-size studios.

**Interviews and Focus Groups**

Interviews and focus groups are scenarios where user researchers sit with users and discuss the product they are evaluating. These can be structured interviews where the user researcher follows a rigid script, unstructured interviews where the interview is more of a freeform conversation with a general topic, or a semi-structured interview, which falls somewhere between the two [23]. Interviews and focus groups give

\(^1\)google.ca/forms/
qualitative data. They can provide both behavioural and attitudinal data, and are very effective for small-size development teams.

2.4 Analytics

Another method in GUR and HCI is analytics, which are of special focus in this thesis. This is because they provide a low-cost means of obtaining important and actionable user information which can be used to improve games [30]. Analytics, within the context of user research, essentially refers to an analysis of potentially large-scale datasets of telemetry recorded from a product. This telemetry is generally in the form of events, as they are instantaneous and easy to record [31]. The events are recorded and uploaded to a database. This database is then analyzed in a variety of different ways, depending on the data and kind of analysis needed. There are a number of tools that are available for developers to use to set up metric recording, as well as aiding with analysis.

2.4.1 Telemetry in Games

Telemetry in games specifically is a topic worth pursuing. Data-collection tools built in-game provide value for user researchers, as there is a wealth of potential analytic data. This can clearly be seen in a research paper presented by Gagné et al [32]. The authors of the article perform an analysis of a simple, free-to-play real-time strategy (RTS) game called Pixel Legions. They work alongside the developer to instrument, collect, and analyze telemetry data. They find that the telemetry analysis provides important and useful information with respect to finding game exploits and content skipping. This paper focuses mainly how data visualization techniques can give insight into data. Existing statistical and visualization tools are used to represent and analyze macroscopic issues like attrition rates, length of play sessions, perceived difficulty, etc. This is done with graphs and plots of these same values. Microscopic analysis, which deals with how a player plays the game, is measured through
Pathways: an interactive visualization system created by the authors which records players’ movements through gamespace and time, separate game events, and provides a time slider to scrub through the entire playthrough. A limitation of this system however, is that while telemetry data provides what a player did, it does not provide the context as to why a player acted [33]. One cannot know why a player quits after only a minute of play strictly by looking at telemetry data. This research shows that telemetry analysis may lack context in a user research session.

2.4.2 Example of Analytics use

A reasonable concern with respect to metric analysis is how effective can it be at actually forming conclusions about a given user tester. However, research presented by Loh et al [34] demonstrates that metrics can be used to cluster player groups, showing a distinct difference between expert players and novice players of a game. The authors of the article build off of previous research that differentiated expert vs. novice performances using similarity measures, which analyzes the dissimilarities of a novice’s action sequences and an expert’s. These action sequences were coded using a task-based approach based on the sequence of objectives/tasks completed in-game. The task-based approach was shown to be potentially more effective at differentiating novice vs. expert behaviours than other metrics like time to complete. This study suggests another method to code action sequences of experts and novices by way of a tile-based (Navigational) approach. The differences are compared using Jaccard coefficients to determine similarity. A game-map is divided into grids of different sizes, with player paths and objectives traced and analyzed. The authors test the effect of grid sizes on differentiating between experts and novices using the tile-based and task-based approach. The researchers concluded that for each game there is an optimal grid-size for the most similar novice and expert behaviours, and that both task-based and tile-based action sequence coding approaches are useful for serious games.
analytics. This research shows that analytics can be used to determine information about players, potentially used to find ideal players for user testing sessions, which can help optimize user research evaluation for small-size studios.

2.4.3 Visualizing Data

One of the challenges for user researchers is presenting telemetry data in a meaningful way. A paper by Wallner and Kriglstein [35] demonstrates many methods for visualizing data, to assist in analysis. The authors review literature on visualization-based analysis of game metric data in order to give an overview of the current state of this field of research. The authors demonstrate first that gameplay data or telemetry of any game event for any game can be recorded and stored. However, the current challenge is that there is no current single best solution for visualizing this large data, with 5 methods currently being used: charts and diagrams, heatmaps, movement visualizations, self-organizing maps, and node-link representations. For the purposes of a small-size studio, the easiest visualizations to use of the ones presented are charts and diagrams, as well as heatmaps. These generally do not require as much analytic training to understand. The research also shows that game analytic data can become so large it is incomputable by traditional means, requiring big data analytics techniques [35].

Additionally, gameplay metrics alone provide no context for a player’s behaviour, which would require other methods to contextualize [35]. Research by Mirza-Babaei et al [36] attempts to circumvent this weakness through visualization by presenting a user session alongside the user’s biometric readings to show the physiological context of the data in a “storyboard” of the session. This research was followed by visualizing the biometric information as a meta-data layer superimposed onto the game world itself, allowing for direct analysis [37].
2.4.4 Contextual Telemetry

Understanding that metric analysis lacks contextual information about a user, research presented by Vijaya and Shivakumar [38] shows that it is possible to record the emotional state of a user by recording their galvanic skin response (GSR), or skin conductivity levels (SCL). A subject whose physiological state is to be measured is shown/played a movie clipping, images, or recorded audio signal, to elicit an emotional response. Due to the change in emotion, GSR varies. Adding this subject’s GSR values, with their explicitly stated emotional state, creates a database which can be referenced. With a dataset of 750 physiological signal measurements, the researchers are able to calculate an average GSR value for an emotion and compare any given subject’s physiological measures and recognize their emotional state with an accuracy of above 80% for most emotions. To build on this topic, Nacke [39] suggests using psychophysiological measures like facial electromyography (EMG) signals to provide context to data. While this research is out of reach for small-size independent developers, it illustrates the strength of telemetry, and how many different ways telemetry can be used.

2.4.5 Tools to Assist Analysis

Metric analysis does not necessarily need to be the sole responsibility of data analysis and user researchers. There are examples of in-house analytics visualization tools being created for both internal and external audiences, such as the one reported by Medler et al [40]. The authors of the paper created the tool Data Cracker, with the purpose of analyzing player gameplay behaviour for Dead Space 2 during the Dead Space 2 development. The focus of the paper is to show Data Cracker as a case study, highlighting the challenges and benefits of designing a visual game analytic tool while working with a game team. The tool records game telemetry client-side, such as match start times, player positions, when players equip weapons, total weapon
damage caused, when players are killed, when players respawn, when a team wins, and experience earned. This data is then uploaded to a game data server as raw text where it is stored in a MySQL database, where it is then processed displayed through PHP and jQuery calls. The key takeaway is that the data is now made available through a web browser, not through a spreadsheet analysis application. This means the data and analysis is inherently more accessible, which is a lesson that can be applied to user research methodologies for small-size developers. The data visualizations are more accessible for all members of the development team to view, which can result in actions being taken faster based on the data.

2.5 Mixed-Method User Research

Not only is there value in using individual user testing methods, but these can be combined to a much greater effect. Research presented by Gómez-Maureira et al [41] suggests that combinations of GUR methodologies provide information that would otherwise be unobtainable, and can improve the quality of feedback for a user test. The authors of the article use three Game User Research (GUR) methodologies (user interviews, game metrics, and psychophysiology) to improve a 2D platformer game. The methodologies are being evaluated to see their usefulness for small-scale game development teams. The 2D game is recreated three times based on feedback from combinations of pairs of the different methodologies. The modified games are evaluated by players playing the game, and then filling out the Game Experience Questionnaire (GEQ), which is a survey used to quantify player experience [42]. The researchers conclude that user interviews provide the clearest indications for improvement among the considered methodologies while metrics and biometrics add different types of information that cannot be obtained otherwise. This demonstrates distinct advantage of mixing GUR methodologies.
As an example of using a mixed-method approach, Thomsen et al [43] combine biometric recording and analysis with questionnaires in order to evaluate the onboarding experience of various free-to-play mobile games. The purpose of the research was to identify elements which create an enjoyable onboarding experience, but the research also presents that the levels physiological arousal correlated with the reported overall experience of players.

2.5.1 Analytics in Mixed-Method User Research

As an extension of the previous point, analytics deserve a special mention in mixed-method user research. Lynn [44] suggests that telemetry in games is a powerful supplement to qualitative user research practices. Telemetry should be integrated into development, not just analyzed after the release of a game. Lynn discusses their work on Volition’s Red Faction: Guerilla, stating that the combination of analytics, interviews, and surveys, was extremely helpful.

As an example of combining surveys and analytics: one user tester claimed in a survey that a given level was too hard. Analytics are able to dive deeper into this issue, showing that the user had a large number of deaths in that level [44]. Additionally, as the locations of each of the deaths were recorded, these were visualized on heat maps and shown to designers, as a potential area of concern. The designers may use this information to change the level layout to be not as difficult. Combining these two methods allows for a thorough analysis based on a single comment in a survey.

Automated physiological telemetry gathering systems, such as facial coding, can be presented alongside recorded sessions much like the biometric storyboards discussed by McAllister et al [14]. The facial coding can allow for the extraction of emotional data which helps provide context to analyses. Understanding the emotional high and low points should allow for an optimized analysis.
Smaller development studios should be able to use analytics in a similar manner. Research proposed by Mirza-Babaei et al. [45] show that analytics methods can be used to highlight problem areas of games in development. In a listed example, consider a platformer game, where all death event locations are logged. If one level were to have a disproportionate amount of deaths compared to the other levels, this is cause for closer inspection. If a user researcher were to be performing observation of a recorded session, in an environment without analytics, this would require viewing the entirely of a play session. However, with analytics, the user researcher can focus only on the problem areas [45].

Small-size developers should be able to use analytics in a similar manner. Analytics can provide focus on specific areas for other user research methods. In this manner, analytics helps optimize the GUR process, reducing the workload of user researchers. This can make GUR more affordable and accessible for these small-size developers.

2.6 Considerations for Analytics Tools

The benefits of analytic tools can be seen from the research presented above. There are many different potential analytic solutions available for game developers. For games post launch, Heilbrunn [46] suggests a number of tools to manage key performance indicators (KPIs) of a game. Some solutions mentioned by name are: Bunchball Nitro Analytics, Gigya Gamification Analysis, GameAnalytics, DeltaDNA, GAMEhud, Honeytracks, and Uptracks. Each of these are third-party solutions which typically have licensing fees for developers. These fees may make these solutions out of reach for a small-budgeted, small-size developer.

Academics pursuing analytic tools for research often find themselves needing to create their own solutions, such as PLATO [47]. This is because industry methods, like Microsoft’s Tracking Real-Time User Experience (TRUE) system [16] are often
proprietary, and are not available for general use. This makes pursuing the creation of an in-house analytic solution somewhat difficult for a small-size developer, as there are not many case studies exploring this research topic [48].

2.6.1 A Comparison Framework for Analytic Tools

From the research presented above, small-size developers have several options available to them when considering analytics tools for game evaluation. Developers must consider the benefits of using various third-party tools, or potentially creating an in-house solution for evaluating games during development. The main points of difference that can exist between these possible solutions have been highlighted in this section. These comparison points have been selected with a focus on what methods are available to small-size developers, and how these methods can assist in performing user research. The goal of this section is to highlight a framework that small-size developers can use to help determine what analytic solution may be best for their game in development.

Affordability

The affordability of a solution is one of the most important aspects to consider when selecting an analytic solution. However, developers need to consider more than just the dollar cost of selecting an analytics tool. Obviously, an in-house solution would be the most attractive solution if only the dollar value is considered in terms of affordability, but this is not the case. In-house solutions cost development time to create and maintain. The development time of these solutions effectively becomes a cost, as it is time and resources that are now no longer being spent on the development of the game itself. From a business perspective, low-cost solutions that provide good results are better to use [49].
Sharability

Small-size studios should consider how much they will need to be reporting and sharing their data. Many analysis tools offer some ability to share their presented data, usually in the form of a dashboard. In fact, this sharability is a noted strength of a potential solution [50]. The small-size studio must decide how much it needs to be able to share its data, and how effectively this data can be presented to different members of the studio. This is crucial, as miscommunications between the stakeholders in development can often result in bugs, flaws, and bad player experiences [18]. Therefore, it is important for small-size studios to have good data sharability. This is equally important for larger companies, or companies looking to grow. Kiloo, a game development studio from Denmark that grew from 15 to 45 employees, puts out daily analytics reports for all company stakeholders, ranging from design, marketing, user interface design, and even artists [51].

Data Refresh Rate

Research has shown that analysis no longer needs to be on static, offline datasets [52]. Live data analysis on a dynamic database is quite doable, especially in the case of small-scale developers. During the development process of a game, it is unlikely that there will be an extraordinarily large amount of user data, simply due to a lack of players. Understanding this, there will be less traffic on the telemetry server, allowing for many more requests uploading data, and requesting to see data [53]. However, this is entirely reliant on servers that are not stressed, which may not be guaranteed with third-party analytics solutions [46]. Depending on the needs of the developer, this may be extremely important or not very important.

Custom Data Analysis

An important consideration for what analytics tool to use is how the data is stored, or how data needs to be formatted for that tool. Some tools allow for lots of flexibility in
their analysis, but some tools such as GameAnalytics have much more limited analysis options, being limited to simple arithmetic options [46]. This can be problematic for some small-studios, as some data analysts may prefer tools that have direct SQL access. Different solutions may offer different approaches to the data. As an example, Data Cracker, the analytics tool used for Dead Space 2, populates its visualizations by using AJAX calls to PHP scripts, which retrieve data tables from MySQL [40]. This approach would require much more programming knowledge to implement, so small-size developers must take care to understand their own strengths.

**Visualization Quality**

The visualization quality is paramount to the effectiveness of an analytic framework. Graphical representations make complex game data easier to understand [37]. If a developer cannot understand the data being shown to them, then they cannot effectively use that data to make informed decisions to improve their game. Understanding this, special care must be taken to produce quality visualizations which confer information effectively. Research presented by Salvucci et al [52] suggest that for event-based data, such as game telemetry, the data should be represented in a visualization panel, such as a dashboard. In addition, it is beneficial to show general summaries of information, allowing for developers to “drill-down” [13] to more specific information. This drill-down process is also an optimization method discussed by Chittaro et al [53], allowing for less expensive computational costs by only showing limited amounts of data.

As discussed earlier, the clearest visualizations to use for small-size studios are charts, diagrams, and heatmaps [35]. These are the visualizations that would be most easily interpreted by those lacking GUR training. An ideal solution would be able to effectively visualize game data at a glance, but also allow for more specific, specialized data analysis.
**Customizability**

The customizability of a solution refers to how well it can be customized to a given project. This idea comes from a discussion where it was stated that Electronic Arts’ Skynet analytic solution used for Dragon Age: Origins was not fully accepted by the development team until it was customized to DA:O [13]. In addition, customizability refers to how able a solution is at sending detailed events to the game server for telemetry recording. Some researchers, such as Charest, suggest that if a solution cannot allow custom event information, that solution should not be considered [54].

### 2.7 Summary (Chapter 2)

This chapter has discussed various research into HCI, including a discussion of the current research paradigms of HCI, the various methods used in HCI, and how these methods can be applied to games. The application of HCI methods on games, used to improve gaming experiences, is called Games User Research. The benefits of GUR are presented, as well as the effectiveness of different methods and combinations of methods.

Special care is taken to mention the use of analytics in mixed-method user research, as this can optimize the user research process for small-size developers. Understanding the need for analytics solutions, a framework for comparing analytic solutions is proposed. This framework is presented in the interests of small-size game developers, by proposing key areas of consideration used to compare analytics solutions.
Chapter 3

Exploring Analytics Solutions

3.1 Overview

As discussed earlier, this thesis discusses the contribution of different analytics solutions in games user research for indie studios. This chapter looks at a case example of an event-tracking plugin and data visualization solution. The chapter first details the development of the tool and then discusses how game developers can implement the tool in their development cycle. The pros and cons of such solutions are highlighted and later discussed in Chapter 6 where the thesis compares them to other potential solutions, such as the ones presented in Chapters 4 and 5.

3.2 Introduction

This chapter serves to present a first step into understanding analytics solutions. It is a largely exploratory chapter discussing the creation of an analytic solution, and why this might be considered beneficial for a small-size developer. It is important to note that this project was worked on by a team of 5 total participants: the author of this thesis and 4 undergraduate students from the University of Ontario Institute of Technology’s Game Development program. Of these 4 students, 2 were responsible for
assisting the author in development of the tool, while the other two undergraduates were tasked with testing the tool.

Your Games User Research Tool, or YoGURT, is a simple-to-use event-tracking plugin and data visualization solution built in order to facilitate the testing of games through data collection and analysis. It is designed for games made by small-size studios as it offers an easy and affordable way to run analysis on their game and facilitate data collection. It is also, however, limited in terms of features and analysis it offers.

YoGURT is designed to be able to track base-level and custom events chosen by the developers. Once this data has been gathered it is stored in sessions on an online database for easy user viewing. There are no limits to what the program can track, as long as it is properly integrated into the game. This gives developers lots of flexibility with which items and events to track in their game, which can later help facilitate analysis.

3.3 Motivation

The entire project of YoGURT stems from the idea that independent game developers generally lack the resources required to perform thorough user tests on their games [9]. Understanding this as a base need, the goal was then to create some way in which these developers could have access to a user research solution for evaluation that has a low resource cost. As was presented in earlier findings [35, 55], analytics solutions are particularly useful here, as they can give information about a game product without requiring a large time investment. Setting up analytics in a game project one time may be sufficient for a large duration of the project.

To build off of this notion of a system setup that was sufficient for a long project duration, a user research tool was proposed to help automate and streamline telemetry recording and analysis. The goal was to make a tool that could be used by any
developer at any phase in the development cycle, but with specific care focused on having a simple process that did not require much user research knowledge to use effectively. In the spirit of designing an accessible product, the tool was named **Your Games User Research Tool**, or YoGURT for short.

Following from findings mentioned earlier regarding metric collection [54, 41, 35, 48], it was decided to use event tracking for the purposes of recording metrics. This is effective simply due to the nature of events in game code. Using events allows game information to be uploaded to the game server at the exact moment that information changes. Conversely, it also allows for events to be uploaded asynchronously, by using push-event systems like the one described by Carzaniga et al [56]. An application of asynchronous event pushing in games user research could be sending the recorded events of a research session to a server well after the session is complete, as long as the events are properly timestamped. In systems without events, variables would need to be monitored and logged at regular intervals. For variables that may not change often, this is a waste of computer cycles, and can potentially clutter up a database, making it harder to analyze for actionable results.

However, overly simplistic tools may not be effective. For many developers, the ideal tool may also need to allow for flexibility and customization in the design. Zoeller points out that the in-house tool created for EA’s *Dragon Age: Origins* team, *Skynet*, was better embraced by the development team as it was customized more to the team’s specific needs [57]. YoGURT’s event-driven metric collection therefore needs to be as simple or as complex as a developer desires. To allow for this, each event could be recorded as a plaintext string. If developers want to simply record events without additional customization, this can be completed as simply as logging the string “death” every time a player dies in a level. Given time resources for a developer, this may actually be sufficient. However, if the developers are willing to manipulate strings for more information, and have the development time to customize the tool,
they could record something more substantial, like “death_xposition_yposition” for a more complex interpretation later on.

One must understand that there are ultimately two halves in an analytics solution for GUR: (1) metric recording and (2) analysis or visualization. YoGURT therefore needs to also be created in two parts, with one aspect focusing on facilitating the recording of game events, and with the other aspect streamlining the storage and visualization of these recorded metrics.

3.4 Developing YoGURT

YoGURT is essentially a two-part project, the technical aspect of which was worked on for a period of roughly one full month by three programmers. The first part of YoGURT, the application plugin, handles recording and uploading metrics to a web server. The second part of YoGURT is the web server and visualization webpage where all the metrics that are recorded can be viewed.

3.4.1 YoGURT Plugin

The YoGURT application plugin is a DLL and code package that can be easily added to any code project. It automates the process of sending messages to a server, and defines a schema by which event information can be sent.

The YoGURT plugin was developed in C++ on a Windows system. This is because C++ allows for the creation of Dynamic Link Libraries, or DLLs, which are plugins which can be loaded and used by other programs very easily. The DLL is quite simple in how it works; it waits for developers to send a custom event to the server, and then builds a JSON object out of the information which will populate the server. In order to handle sending requests from C++ code to a web server, the open-source library HappyHTTP was used [58]. To pack the data into a JSON object, the open-source library RapidJSON was used [59].
Table 3.1: Common Parameters in Analytics Solutions

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Additional Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Name</td>
<td>Indicate which action happened</td>
</tr>
<tr>
<td>User ID</td>
<td>Identify who performed the action</td>
</tr>
<tr>
<td>Timestamp</td>
<td>Indicate when the action was performed</td>
</tr>
<tr>
<td>Session ID</td>
<td>Uniquely identify the play session for the user</td>
</tr>
<tr>
<td>Build Version</td>
<td>Indicate the current build version</td>
</tr>
<tr>
<td>Platform</td>
<td>iOS, Android, Windows, Amazon</td>
</tr>
<tr>
<td>Device / OS / Hardware</td>
<td>Technical details that are important to the game</td>
</tr>
<tr>
<td>Additional Event Parameters</td>
<td>Indicate any additional parameter specific to the event</td>
</tr>
</tbody>
</table>

In research presented by Charest [54], all in-game events should follow a similar structure. Listed are some events parameters that would be common to any analytics solution:

For the purposes of hastening development, these parameters were consolidated into simpler ones. Most notably, the **Platform** and **Device/OS/Hardware** parameters. What was considered to be the bare minimum information that should be recorded for an event were the event’s (1) gameID, (2) buildID, (3) instanceID, (4) sessionID, (5) event string, and (6) time of event.

The *gameID* is the unique identifier of the game being created. While not directly stated in the table presented by Charest, each game in development should have a unique identifier, as developers might be working on several games at a time.

The *buildID* represents the particular version, or build, of that game. This is especially important as it helps understand which build of a game is being tested. Development is often iterative, and telemetry from a previous build may no longer be relevant or accurate. Additionally, several builds of a game might be tested concurrently, requiring this parameter for differentiation.

The *instanceID* refers to a specific installation, or instance, of the build. This is useful to have as it is unique per installation, theoretically allowing for per-user analysis. It could be used to find issues on different hardware, as all events are
tagged with the installation. It is somewhat a consolidation of the **Platform** and **Device/OS/Hardware** parameters, but on a much more limited scale.

The *sessionID* is a unique identifier used to help differentiate play sessions on a given installation, allowing for multiple datasets on the same instance.

The *event string* is the string that holds the information of an event that the developers wish to track. This is the most important property of the event to track, as this is the property developers are allowed to specify or customize in code. It is a consolidation of the **Event Name** and **Additional Event Parameters** presented in the table above. At its most basic level, the event string could be simply the name of an event, with no additional information. However, the event string could also theoretically be a more complex JSON object formatted as a string, with more event parameters specified. This simplification allows for YoGURT to be included in a development project faster, while allowing for deeper customization.

The *time* is simply the unix timestamp of when the event occurred.

All of the properties specified above are encoded in a JSON object, and sent to the server to be processed. This is a sample event object:

```json
{
   "gameID": "Demo Game",
   "buildID": "0.0.1",
   "instanceID": "a3ba8924-93a2-411a-ab75-56331d840bc6",
   "sessionID": "1",
   "event": "player_jump",
   "time": 1495788227046
}
```
3.4.2 YoGURT Web Server

The YoGURT web server was created as an easily accessible location where all recorded metrics may be viewed. It was designed to allow for “drilling-down” from general information to more specific information, similar to EA’s Skynet [57]. This drill-down approach allows for organization in displaying multiple different games, installations, builds, and sessions of play. It was developed using a Windows, Apache, MySQL, and PHP (WAMP) stack. It uses a MySQL database to hold tables full of events that have been logged. PHP code is used to interpret the JSON data as sent by the YoGURT plugin, where it is used to populate the database.

The PHP scripts procedurally create tables in the database for each game, instance, and session. The database structuring is roughly as follows:

- There is a table that contains a list of all of the different games, or gameIDs

- Each individual game has a table that contains all of the different versions of that game, or instanceIDs. This table is named the same as the gameID (eg: “MyGame”)

- Each instance of a game has a table that contains all of the different sessions played of that instance of the game, or sessionIDs. This table is named as the gameID, plus the index of the instance (eg: “MyGame_1” or “MyGame_2”)

- Each session of a game that has been played has a table containing all the events that occurred during that session. This table is named the same as the instance, plus the index of the session (eg: “MyGame_1_1”, “MyGame_1_2”, “MyGame_2_1”, etc)

See Figure 3.1 for a visual representation of the database tables. This nested, procedural table creation is somewhat inefficient, simply because it is a somewhat confusing design pattern. It would perhaps be less confusing to have a single but larger
Figure 3.1: Rough database diagram. The *games* table is populated with all games, like *first_game* and *second_game*. Each game has a number of instances, and a table for each of these instances, such as *first_game_1*. Each of these instances has a table of all sessions, like *first_game_1_1*, with each session table full of the event data per session.

Table full of event, session, and instance data per game. Once full of information, the tables are queried with PHP scripts. The results of these queries are displayed using Javascript and jQuery on a front end webpage. In addition to displaying data tables, the front end webpage also displays a simple visualization of all the events in a bar graph, showing which events were the most frequent. This visualization was created using Google Charts\(^1\).

\(^{1}\text{developers.google.com/chart}\)
3.5 Using YoGURT

Using YoGURT is a simple process that does not require much background user research knowledge to use effectively. It is a plugin built into a DLL, so that all a developer has to do is unpack the plugin, include it in the game project (which can be a C++ or Unity project, as examples), initialize the tool with server information, and start logging events. The event logging process is very straightforward, it is a simple function call to the YoGURT api:

```cpp
YGURT::customEvent("custom_event_name");
```

That is all it takes to log a custom event once the project has been set up. The server collects all of these event calls and then stores them by their event name. As an example, any time an event named “death” is logged, it will be stored as the same event type on the server. This allows for the same event call to be placed in multiple different locations within the code of the game project. This is all a developer needs to do in their game project in order to collect data for YoGURT. As the events list is populated they are logged on the YoGURT site where developers can view all the data from each individual session with great ease.

Server-side, the game data is stored in a MySQL database as designed by the YoGURT api (this database construction is also automated). Once the user navigates to their YoGURT installation in any web browser, they are greeted with a list of all the games they are recording data for. YoGURT was made with developers working on potentially many games in mind.
Each game is then broken down into individual instances of the game, where an instance represents an individual user’s installation of the game. As more players play the games, more instances are created and added to the server database. The instance name is a randomly generated string of letters and numbers to ensure anonymity of data.

These instances are then broken down into individual sessions that the user played, as a user might play more than one session on their given game installation. This can tell a developer who is frequently starting and playing the game, which could indicate either that a user particularly enjoys the game, and is launching it frequently, or perhaps they are having issues in their installation and frequently have to restart the game. This could be determined by looking at event timestamp data in the list of events. Each session is also given a randomized string name.
Sessions

<table>
<thead>
<tr>
<th>Session1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session2</td>
</tr>
<tr>
<td>Session3</td>
</tr>
<tr>
<td>Session4</td>
</tr>
</tbody>
</table>

Figure 3.4: Sample list of all game sessions of a given instance in the YoGURT database

Each session contains the list of event data that occurred during a given gameplay session. It tells the developer the actions that the user performed, also providing information about the order of these events during gameplay, by organizing them chronologically and by providing timestamps of these events.

Events

<table>
<thead>
<tr>
<th></th>
<th>Event</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>session_start</td>
<td>2017-05-25T19:47:51Z</td>
</tr>
<tr>
<td>2</td>
<td>start lvl 1</td>
<td>2017-05-25T19:47:57Z</td>
</tr>
<tr>
<td>3</td>
<td>jump</td>
<td>2017-05-25T19:47:59Z</td>
</tr>
<tr>
<td>4</td>
<td>jump</td>
<td>2017-05-25T19:48:02Z</td>
</tr>
<tr>
<td>5</td>
<td>jump</td>
<td>2017-05-25T19:48:04Z</td>
</tr>
<tr>
<td>6</td>
<td>jump</td>
<td>2017-05-25T19:48:06Z</td>
</tr>
</tbody>
</table>

Figure 3.5: Sample list of all events in a given game session in the YoGURT database

The server also breaks down the events logged into a chart visualization (see Figure 3.5) to tell the developer how often certain actions are being performed. This can be used to determine several valuable pieces of information such as what item is most
used, or what locations the user visits most often. What events a developer tracks in the game and visualizes in the server is ultimately up to them.

![Event counts graph](Figure 3.6: Graph showing counts of events in a given game session in the YoGURT database)

### 3.5.1 Usage Scenario

As a sample usage of the event visualization, one can imagine a scenario where a developer wants to measure if ranged combat in a game is more effective than melee combat. They would then log each enemy death event with either “killed_by_range” or “killed_by_melee”. Their game designer has decided that overall these should have equal risk/reward, so over a long play session the designer hopes that these numbers are near even. If there is a large disparity, for example, there are significantly more ranged kills than melee kills, then melee combat may not be rewarding enough. Or perhaps, melee combat is perceived as too dangerous for the tester. This information has highlighted that there might be a problem in the balance of ranged vs melee combat, and as such makes it much easier for a designer to identify this.

In a user research scenario of the above hypothetical game, having the information of a player’s preference towards ranged/melee can help guide interviews or other GUR
methodologies. This is an example of using analytics methods in tandem with other methods, demonstrating the benefits of a mixed-method approach [60].

3.6 Discussion

From the comparison framework presented at the result of findings from Chapter 2, one can see that the key points of analysis are (1) affordability, (2) shareability, (3) data refresh rate, (4) custom data analysis, (5) visualization quality, and (6) customizability. YoGURT will now be evaluated based on these criteria.

Affordability: As an in-house solution, YoGURT is quite affordable. Other small-size studios should be able to recreate a similar tool using similar resources. Open-source libraries make this attractive and affordable. However, there was a month of development time on the project, with 3 developers working on it. This is an important consideration for developers who may require a user research solution in a shorter time.

Shareability: One of the key goals of YoGURT is accessibility for users, and making information available on a simple webpage meets this goal. It also conveniently makes it accessible by many other, providing an easy, shareable link to the information. In this regard, the web client approach of YoGURT makes is perform admirably with respect to shareability.

Data Refresh Rate: Data refreshes in the YoGURT web client as the webpage refreshes. The structure of the webpage is designed to better show completed user tests, instead of ones that are currently in progress. However, in a more customized web client, the server could be polled multiple times in a live user test, so the only limitation of this speed is the hosting server. In this regard, YoGURT and any other potential in-house solution do quite well.

Custom Data Analysis: Outside of presenting the event tables in the web client, YoGURT does not offer much in the way of custom data analysis. The PHP scripts
pull data directly from the SQL database, but YoGURT offers no way to use that data outside of simply presenting it. This is a distinct weakness of this tool.

Visualization Quality: Unfortunately, this is a reasonably large limitation of YoGURT. The event count graph (Figure 3.6) can only present so much information, and does not have the ability to show any form of event tracking over time. This makes the visualization of YoGURT specifically lackluster. However, with enough customization to the web client and server, better visualizations could be achieved. Another negative in this regard is that YoGURT does not have a chart builder, making it difficult for non-programmers to make visualizations.

Customizability: As an in-house tool made from open-source libraries, YoGURT is infinitely customizable, as could be said of any in-house solution. It is also reasonably usable without customization, so in this regard it performs quite adequately.

3.7 Summary (Chapter 3)

YoGURT is a simple to use GUR tool made with the goal of facilitating user research. It is composed of two parts: a plugin which helps with the recording of game telemetry, and a web server which displays the data. The ease of using the tool paired with the fact that it only really needs to be set up once helps it be an effective, low-cost tool for the small-size developers. This event-driven solution allows for enough flexibility in its design that it can be effective at its bare minimum, while allowing for users to gain more potential out of it by customizing the tool.

This initial exploration into potential analytics solutions shows promise. It shows that with some time investment, small-size game developers do have access to forms of user research. Even the minimal example of YoGURT has benefits to use, and can be used as a reference guide for developers creating their own in-house analytics solutions.
Chapter 4

In-House Solution

4.1 Overview

Following the work of the initial development of YoGURT, an attempt at an in-house solution to mimic what unfamiliar developers might create, it was found that the visualizations were somewhat lackluster. Additionally, due to the way the information was presented and structured, it could not be used effectively for live datasets unless it were customized. Stemming from these shortcomings, a tool was designed to show multiple visualizations, pulling data from a live dataset to be used to assist with live user test reporting. The tool, Autodash, was tested in a case study, and used in a live playtest report.

4.2 Introduction

The previous chapter discussed the development of YoGURT, a simple analytics solution as the initial exploration of possible analytics solutions for indie studios. One notable critique of the tool is that the visualization of the data is nothing more than a frequency distribution plot. While this does have uses, it is quite limited in that it does not allow for multiple visualizations for a given dataset, nor does it allow much in the way of building these custom visualizations. While YoGURT is technically
customizable to the point where it does these things, it does not have that potential in its current state.

To follow this first attempt analytics tool, a tool featuring much better visualization for the purposes of rapid analysis was created. This tool, known as Autodash, uses stored procedures from a SQL database which produce formatted data, which is then represented in various charts and tables. This dashboard is fully customizable, with the ability to add different charts and tables based on given data sources. This makes data analysis faster and easier for developers, resulting in more optimized user tests.

Autodash was created during the author’s internship placement. It was created to automate and expedite the visualization of user testing data shortly after a testing session. This allowed for actionable results to be drawn from the data in a much shorter time than if it had been analyzed and graphed by hand.

Disclaimer: As much of the content of this chapter was explored during a private internship on a protected intellectual property under development, some of the information has been withheld.

4.3 Motivation

As Autodash was created during an internship of the author’s, with the intent of solving problems specific to that company. The internship was undertaken in a small-size gaming studio over a portion of the summer of 2016, and was done to gain practical insight into analytic processes used in the gaming industry. By working directly alongside a small-sized studio, this particular solution was developed with their needs in mind.

4.3.1 Studio Needs

The author assumed the role of a data analytics intern, responsible for the creation of an analytic tool (see section 4.4) to help automate playtest reporting. This au-
tomated playtest reporting would be used to evaluate an in-development game (see 4.3.2). Working directly with small-size studios as a data analyst allows for better understanding of the user research needs of these studios.

4.3.2 Stream Game

The game being developed at the studio was a group-based dungeon crawler in a rich environment. What makes the game unique is that it is a game broadcasted on a live stream, that could be interacted with by viewers. The game being broadcasted will be referred to in this chapter as the broadcasted game, and players of this game will be referred to as the players.

Viewers of the live stream can interact with the broadcasted game, by managing and spending currencies on items. These items manifest in the broadcasted gameworld, affecting the players and changing the viewing experience. This viewer interaction was crafted to be a spectator game in its own right. For the purposes of this chapter, the viewer interaction will be referred to simply as the viewer game, and those who are watching and interacting with the stream will be referred to as spectator-players.

The game encourages active participation of spectator-players by rewarding those who were watching and engaging with the broadcast. The rewards came in the form of currencies, which could be used in turn to further engage with the broadcast. From a design standpoint, it became important to gauge and scale the amount of currency inbound and the amount of currency sinks available to the spectator-players. These are metrics that are easily recordable in a database, making an analytics solution very attractive. Additionally, due to the live nature of the game, having a live representation or dashboard of these analytics would be extremely helpful for analysis.
4.3.3 Analytics Pipeline

To understand the necessity of Autodash, one must understand the methods by which analytics were processed for this game during development. To complement user testing methods, metrics relating to the purchases made in the viewer game are recorded and stored in a SQL database.

After a typical user testing session, the analyst would query the database and prepare tables of data, formatted in a variety of ways to show spending habits of players. Example charts could be: breakdowns of items purchased, average currency spent on a transaction, what the most popular items were, and a demonstration of currencies being taken out of the system and being put into the system. Each of these visualizations would then be put into a presentation and displayed for the designers to prioritize the next focus of development.

These user testing presentations, or reports, provided actionable results. But between analyzing the server data, formatting and visualizing the charts, and compiling all the information into a single report, this process often took days to complete. Additionally, if the main analyst were absent, that would also slow the process, as few others would be able to analyze and graph the data. Presenting solutions to these problems is one of the goals of this thesis, to reduce the amount of time required to perform analysis, and to make user research more accessible for the developers.

4.3.4 Automated Reports

From the above sections, there was displayed a need for a solution which could speed up the process of analyzing the game data, and also visualizing it. A system of automating the analytics process would solve these two large problems. To effectively replace a report, which would have multiple visualizations and analyses, there would need to be a solution which could effectively represent these multitudes of charts.
An automated report would help these small-size developers by reducing the time to create a report, making user testing more time-effective during development.

As is common in many business solutions tracking metrics [50], a dashboard was proposed, as it could easily fulfill the requirements above. A dashboard could have multiple visualizations on a single page, with the creation of these visualizations being automated thanks to tools like Google Charts\(^1\).

### 4.3.5 Requirements

After understanding that a dashboard would be the solution to assist in automating a user research report, the analytics team needed to decide on which solutions were available to them. Remembering the comparison framework presented in Chapter 2, the dashboards would be compared based on the following criteria:

1. **Affordability**
   - Based on budget, ideally looking for a more affordable solution
   - Based on contract, development and implementation should take between 1-2 months

2. **Sharability**
   - How easily can multiple users be accessing a given dashboard?
   - There must be a minimum of 5 concurrent users viewing the dashboard

3. **Data refresh rate**
   - For this, live game, it must be near real-time (at least under one minute)

4. **Way to view custom data, one of:**
   - Direct MySQL connection
   - JSON Parsing
   - CSV Parsing

5. **Visualization quality, or the quality of the charts represented**
   - Is the dashboard consistently shaped?

\(^1\)developers.google.com/charts
• Are the axes legible?

6. Ability to customize the service

• How much freedom is given with HTML or source access?

Several dashboarding solutions were found which might meet these requirements, such as Klipfolio\(^2\), Cyfe\(^3\), Freeboard\(^4\), Geckoboard\(^5\), and Finalboard\(^6\). Each of these solutions were compared and contrasted based on all of the key metrics. This comparison can be seen in Table 4.1.

As can be seen from Table 4.1, all the solutions offered are competitive. Of all of these solutions however, Finalboard was chosen. The largest contributing factor to this decision was the fact that Finalboard was tied for lowest data refresh rate, as low as 1 second. In the live broadcasted game, for the purposes of user research, the fastest possible analytics were required. Thus, this point of comparison is weighted more heavily than some of the others, resulting in two real competitors. Finalboard’s strength over the other solution with low refresh rate, Freeboard, is that Freeboard suffered from poorer visualizations, especially with respect to its inconsistent layout with respect to multiple shared users. Finalboard offers a much more consistent solution with attractive visuals, making it easier to share the dashboard and report on evaluations. This helps the evaluation process for small-size studios during development.

\(^2\)klipfolio.com  
\(^3\)cyfe.com  
\(^4\)freeboard.io  
\(^5\)geckoboard.com  
\(^6\)finalboard.com
Table 4.1: Comparison of dashboarding tools

<table>
<thead>
<tr>
<th>Metric</th>
<th>Klipfolio</th>
<th>Cyfe</th>
<th>Freeboard</th>
<th>Geckoboard</th>
<th>FinalBoard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affordability</td>
<td>$24/month</td>
<td>$19/month</td>
<td>Free, or</td>
<td>$24/month</td>
<td>$49, once</td>
</tr>
<tr>
<td></td>
<td>5 Users</td>
<td>10 Dashboards</td>
<td>$12/month</td>
<td>1 User</td>
<td>Free trial version with seemingly complete functionality</td>
</tr>
<tr>
<td></td>
<td>10 Dashboards</td>
<td>Public links</td>
<td>for 5 Private dashboards</td>
<td>1 Dashboard</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 Private link</td>
<td>Free plan only has 5 widgets across all dashboards</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sharability</td>
<td>5 Users with edit permissions</td>
<td>Unlimited users once paid</td>
<td>Unlimited readonly shareability</td>
<td>Base of 1 User, 1 Dashboard</td>
<td>Limited by hosting server only (Unlimited)</td>
</tr>
<tr>
<td></td>
<td>1000 may view a public link</td>
<td>No sharing unless paid</td>
<td>Dashboard layout inconsistent across shared users</td>
<td>Up to 5 of both</td>
<td></td>
</tr>
<tr>
<td>Data Refresh Rate</td>
<td>As low as 1 minute</td>
<td>As low as 1 minute</td>
<td>As low as 1 second</td>
<td>As low as 30 seconds</td>
<td>As low as 1 second</td>
</tr>
<tr>
<td>Visualization Quality</td>
<td>Good, consistent, visualizations Legible axes</td>
<td>Good, consistent, visualizations Legible axes</td>
<td>Fairly poor visualizations Ineffective axis labelling Becomes disorganized when sharing links</td>
<td>Good, consistent, visualizations Legible axes</td>
<td>Good, consistent, visualizations Legible axes</td>
</tr>
<tr>
<td>Ability to Customize</td>
<td>Raw HTML Access, potentially limitless</td>
<td>iFrame access, potentially limitless</td>
<td>Raw HTML Access, potentially limitless</td>
<td>None</td>
<td>Full customizability</td>
</tr>
</tbody>
</table>
4.4 Development

Finalboard is a dashboarding solution that requires it be hosted on a web server, as it relies on javascript libraries, particularly AngularJS. The Free Trial version of Finalboard was the version that was chosen to be used, as it is a completely functioning dashboard. The only drawbacks of this are that it has obfuscated source code, and no developer support. However, for the purposes of research and testing different methodologies, this was sufficient.

4.4.1 Widgets

The Finalboard API uses a javascript file to initialize a given dashboard. Each chart or table that one would want to visualize in the dashboard is called a widget. These widgets are JSON objects, containing information about the properties of that widget. As an example of a widget, please see Figure 4.1.

The JSON object containing the information that initializes the widget presented in Figure 4.1 would look like this:

```json
{
    "template": "widgetDirectory/clock/clock.html",
    "js": "widgetDirectory/clock/localtime.js"
}
```

The template property represents the HTML file that hosts the container to the widget. The most important thing in this HTML file is the <div> tag which gives the widget space a unique name or identifier. The js property of the JSON data

![Figure 4.1: Sample widget showing the local time](image)

Figure 4.1: Sample widget showing the local time
Table 4.2: Properties of the Autodash widgets

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>Title of the widget</td>
</tr>
<tr>
<td></td>
<td>Appears in the widget header</td>
</tr>
<tr>
<td>tag</td>
<td>Unique HTML tag for the widget</td>
</tr>
<tr>
<td></td>
<td>Injected as a &lt;div&gt; object</td>
</tr>
<tr>
<td>source</td>
<td>The data source</td>
</tr>
<tr>
<td></td>
<td>Path to PHP script which populates chart data</td>
</tr>
<tr>
<td></td>
<td>The PHP script calls a SQL stored procedure and formats the result in JSON</td>
</tr>
<tr>
<td>chart_type</td>
<td>The type of Google chart if using one (pie, bar, column)</td>
</tr>
<tr>
<td>chart_options</td>
<td>Path to a JSON file containing the default Google chart options</td>
</tr>
<tr>
<td>custom_options</td>
<td>Custom chart options that can be set per each Google chart</td>
</tr>
<tr>
<td></td>
<td>Can be done on individual basis</td>
</tr>
<tr>
<td>export_btn</td>
<td>True/False value</td>
</tr>
<tr>
<td></td>
<td>If there is to be an export button on the widget</td>
</tr>
<tr>
<td></td>
<td>This creates a CSV file out of the chart data</td>
</tr>
<tr>
<td>refresh</td>
<td>Refresh rate of the widget, in milliseconds (ms)</td>
</tr>
<tr>
<td></td>
<td>Interestingly, this was actually in the default Finalboard, but did not work</td>
</tr>
<tr>
<td></td>
<td>properly for integrated charts until repaired</td>
</tr>
</tbody>
</table>

represents the javascript file which updates the clock. It does this by updating the contents of the <div> tag as specified in the template file. These template and js properties are some that were defined in the initial Finalboard release, but they did not allow for enough data customization. As a result, several more properties were created, as presented in Table 4.2.

Based on these additional properties, a more complex widget would understandably have a more complex JSON object. For example, consider a Google Chart which might show all of a spectator-players’s interactions over time for the duration of the broadcast. Such a chart can be seen in this chapter (Figure 4.4). The JSON object for this visualization would appear as such:

```json
1 {
2     "title": "Spectator Interactions Over Time",
3     "tag": "spectator_interactions_over_time",
4     "source": "scripts/getSpectatorInteractionByTime.php",
```
"chart_type": "line",
"chart_options": configBase + "googleCharts/default_options.json",
"custom_options": {},
"refresh": 0,
"export_btn": true,
"template": configBase + "googleCharts/chart.html",
"js": configBase + "googleCharts/chart_handler.js"

4.4.2 Data Sources

The source property of the widgets is a path to a PHP script which populates chart data. These scripts query the SQL database and format the data into JSON, so that it can be used in the widgets. The queries that are called are all predefined stored procedures, which do most of the hard work when querying the database. Stored procedures are used because they reduce the amount of network traffic going to the server, have faster execution, and allow for better database security (by allowing EXECUTE access to the dashboard account but disallowing INSERT/DELETE access).

Many of the stored procedures perform complex combinations of queries in order to present data in a meaningful way. Pivoting data tables was of particular difficulty, as the MySQL server holding the data did not have a pivot operation. Typically, pivoting data tables is not really a job handled by SQL, as spreadsheet tools like Microsoft Excel are made to perform this task. However, to automate the data representation, it would not be sufficient to require a person to use Excel just to format a table. The workaround created was adapted from a solution found online by Stack Overflow user Bjoern [61], where a SQL query is built by concatenating strings together, and the contents of these strings are built from other queries. The
completed concatenated string is then prepared and executed in the stored procedure, producing a formatted result returned to the PHP script.

The result of the stored procedure is a data array, which then needs to be processed. Depending on whether the data needs to be formatted for a Google Chart or for HTML Table insertion, it must be slightly different. Regardless of how the data is formatted, it is output as the result of the PHP page. The content type of the page is also changed to be a JSON document, allowing HTTP GET requests to properly process the data.

4.4.3 Visualization

After retrieving the data, it must be visualized. All of the visualization of the dashboard is done using one of two methods: a Google chart, or an HTML table for text data. Both methods make use of the customized dashboard parameters and require that the data to fill the visualization is in JSON format. These formats are slightly different from each other, but that is a result of the difference in how Google Charts requires formatting and how AngularJS requires formatting.

**Google Charts Visualization**

The Google Charts visualizations use the Google Charts API. These charts are created by using AngularJS to inject a unique `<div>` tag into a template HTML file, and then using Javascript to change the content of that `<div>` tag, populating it with a chart.
In order to get the data to fill these charts, an HTTP request is made to a PHP script on the server, which formats the data in the SQL server, and returns a JSON string. The JSON string is interpreted by the Google Chart and formatted into a graph. Visualizations like graphs are easy to understand at a moment’s glance, making them effective for the quick user test sessions that might be needed for small-size studios during development.

HTML Table Visualization

Similarly to the previous section, the HTML tables are created by using AngularJS to inject a unique <div> tag into a template HTML file. However, where these tables differ from the Google Charts widgets is that there is no need for an additional Javascript file to further process the information, as the AngularJS directive ng-repeat can iterate through the data source and populate a table element.
As was the case with the Google Chart visualization, an HTTP request is made to the source property of the widget. This is a PHP script which formats SQL server data, and returns it as a JSON string. This JSON string is interpreted and formatted as an HTML table. These data tables are not as easy to understand at first glance compared to graphs and charts, but they can be used to provide important contextual data to accompany the graphs. The combination of these two methods can be quite effective for small-size studios performing user research during development.

4.5 Case Study

As described in the motivations portion of this chapter, section 4.3, the goal of the dashboard was to assist in automating and quickening the creation of user research reports. The dashboard did not undergo a formal evaluation, but was used in conjunction with a user test to validate the effectiveness of the tool. The focus of this case study is not to demonstrate how using user feedback improved the viewer game, but instead to demonstrate how a small-size team of developers could use Autodash to facilitate user testing.

4.5.1 User Test

The user test session was run with 5 participants acting as spectator-players. There were additional viewers participating, but these were not counted in the user testing session. Each tester was told to watch the broadcast stream, and interact with it however they felt was appropriate. All of their interactions in the viewing client were recorded in the database.

The goal of this test was to gauge the amount of currency inbound and the amount of currency sinks available to the spectator-players. Additionally, there was a desire to learn about the spending habits of players. A report created after this user test should be able to present breakdowns of items purchased, average currency spent on
a transaction, what the most popular items were, and a demonstration of currencies being taken out of the system and being put into the system.

To meet the goals of this test, an event was created which would track all the occasions where a spectator-player’s currency would change, effectively being a transaction. These events would track:

1. The amount of currency being exchanged (positive or negative)
2. The source of the currency (if positive change)
3. The sink of the currency or where it was spent (if negative change)
4. Special information regarding item purchases, such as the name of item

All of these transaction parameters can be used to provide analysis information to help meet the goals of the user test.

4.5.2 Dashboard Setup

To ensure a smooth process, the Autodash and all of its widgets were set up and created beforehand, including the data sources and SQL scripts. This was accomplished using the methods described in the development portion of this chapter, section 4.4. Google charts were created which would process JSON data output by PHP scripts, and these PHP scripts would be outputting formatted SQL queries. These SQL queries were created as stored procedures to format database information into pivot tables [61]. After the setup time, no additional work needs to be done in order to maintain the dashboard.

Spectator Interactions

The Spectator Interactions chart displayed the number of interactions a viewer made over the course of the broadcast. An “interaction” is any action that the viewer could take to participate in the viewer game, such as purchasing an item to affect the broadcast.
Spectator Interactions by Type

To complement the previous chart, each of the interactions that a viewer took were additionally separated by the type of interaction. Now the specific actions of interest of a user can be assessed, instead of simply consider their total interactions over the course of the broadcast.

Average Tickets per Spectator

The average tickets per spectator chart is a measure of the amount of currency each viewer had over time. Understanding that using currency is how viewers interact with the broadcast, this chart can help bring context to a viewer’s spending habits and number of interactions.
4.5.3 Automated Report

Understanding the needs as presented in the Motivations portion of this chapter (section 4.3), having graphs and tables ready-made for the developers to view at any time was a bonus. The analyst did not have to take time after the end of the testing session to hurriedly compile all of the results into charts and validate the data; this process was automated. It allowed for an immediate debrief of the testing session.

As an additional bonus, the dashboard was very accessible, able to be viewed on any computer with internet access. Typically, the analysis would be done in Excel, and screenshots of the graphs would need to be made, as not everyone in the development team would have access to Excel. However, this dashboard was available online with no need for a spreadsheet tool, so it was available for all to see.
Autodash provided a quick visualization of the data for the developers. The data presented in the dashboard could even be used in a more formal report of the playtest, as all of the information present in any of the charts can be exported as a CSV file.

4.5.4 Using Autodash

Autodash was able to show a series of visualizations which were effectively used as an automated report. This automated report was used to make quick, general feedback about the user test session which was backed by data. In the quick stand-up meeting with developers that occurred after the user test, some general comments could be made about the state of the game.

For example, consider the chart which displays the amount of user currency over time (Figure 4.6). This chart shows that spectator-players had roughly the same amount of currency from the 18th to 30th minute of the broadcast. They were not gaining any income, showing that not much action was happening in the broadcast. This also shows that they were not spending any currency to affect the broadcast either. This is confirmed with the chart that represents the amount of user interactions that occurred over time (Figure 4.4), showing the same null of spectator-player
participation between the 18th and 30th minute. This information allows the developers to focus in on that period of the broadcast, to see why spectator-players were not participating.

Pairing Autodash’s automated reporting with other GUR methods can increase the efficiency with which evaluations are performed. Take the example above, of a user researcher watching a spectator-player interact with the broadcast. Normally, this observation would take at least an hour, the length of the broadcast. However, with the dashboard presenting the lull in the broadcast between the 18th and 30th minutes, a user researcher can prioritize that 12 minute segment of the broadcast to get results sooner.

By allowing a user researcher to focus in on that 12 minute segment, Autodash is able to help prioritize critical fixes in a user research session. In this manner, it optimizes user tests, becoming particularly helpful for small-size developers to gain insight into their games in development.

4.6 Discussion

Autodash served its purpose very well, by creating a very accessible means of visualizing the analytic data, simply by allowing users to view a webpage with the data analyzed and charted. Its biggest contribution was the automation of the analytic process, reducing the time between user testing session and discussion. However, actually drawing conclusions from the charts and tables was still a time-consuming process requiring analysts. As the creator of these widgets, data sources, and stored procedures, it was easy to see the trends and relations between the charts. Those who were not as acquainted with the analytic process were not able to immediately glean the relevant information from the charts. This does not say the dashboard is a failure though, it still has its use in that it automates the visualization of data. Even though
a data analyst or user researcher is still required to draw meaningful conclusions from
the data, once setup, they can be easily trained to interpret the data.

An unfortunate aspect of Autodash that must be mentioned however, is that
without programming knowledge, it is a difficult tool to use and create dashboards
for. While seasoned analysts may be familiar with SQL and stored procedures, they
may lack the knowledge to make PHP scripts or custom Javascript methods to parse
the data and represent it in the chart. Additionally, Autodash used JSON to store and
initialize widget properties, and creating a chart by manually typing in the properties
was confusing to non-programmers. This could perhaps be alleviated with a graphical
user interface in future work, but this would increase the development time for any
developer attempting to make a similar tool.

A special mention of the dashboard however, is that due to how quickly the data
is visualized and presented to the users, it could be used by a user researcher to gain
insight into a tester’s session, before even starting an interview or a focus group with
that user. In this regard, the dashboard could be seen as invaluable, by supplementing
existing GUR methodologies with its real-time analytic reporting. This could allow
for more targeted feedback from a user, as an interview could be tailored to be more
specific as a result of the instantly reported data.

4.6.1 Comparison Framework

After having shown a use case of Autodash, it can now be assessed based on the
comparison framework presented in Chapter 2 (section 2.6.1). The key points of this
analysis are (1) affordability, (2) shareability, (3) data refresh rate, (4) custom data
analysis, (5) visualization quality, and (6) customizability.

Affordability: As an in-house solution, Autodash is quite affordable. In this sce-
nario, the dashboard software used would cost $50 to obtain a license, which is quite
inexpensive. It took a single developer a month of development time to create the
project, which is important to note for small-size studios looking to evaluate the time they can spend building analytics solutions. However, it is also worth noting that Autodash is simply a visualization tool, and does not have a telemetry gathering tool like the solution presented in previous chapter, YoGURT. This would increase the development time, if developers wish to pursue this solution.

*Shareability:* One of the key goals of Autodash was the accessibility for users, and making information available on a simple webpage meets this goal. It also conveniently makes it accessible for many other users, providing an easy, shareable link to the information. In this regard, the web client approach of Autodash makes it perform admirably with respect to shareability.

*Data Refresh Rate:* Data refreshes in Autodash as quickly as specified. For this project, having a fast, live data refresh rate was a very important requirement. In this aspect, Autodash performs quite well.

*Custom Data Analysis:* Autodash relies mostly on premade SQL queries, which disallows most potential room for customization without coding knowledge. However, through the use of PHP scripts which format the data into a usable JSON file, Autodash actually performs quite well with regards to custom data analysis. This does require coding knowledge though. Additionally, Autodash allows for the JSON in the widgets to be exported as a CSV file, which can be used for further processing later.

*Visualization Quality:* The visualizations in Autodash are based on HTML tables and Google charts visualizations, and in this regard they perform quite well. All axes are labeled and the dashboard is visually pleasing. However, a reasonably large negative of Autodash is that it lacks a widget builder, making it difficult for non-programmers to make widgets.

*Customizability:* As an in-house tool made from open-source libraries and a single closed-source project (Finalboard), Autodash is quite customizable to a given deve-
loper’s needs. However, this requires a reasonable amount of knowledge in order to perform.

4.7 Summary (Chapter 4)

Autodash was built with the goal of automating playtest reports. While it did not fully succeed in this manner, it does facilitate the analysis of the data, quickens the analytics pipeline, and presents an interesting case for mixing GUR methodologies to get more time-efficient user feedback. It relies on having an existing telemetry recording tool, so Autodash is not a complete solution.
Chapter 5

Third-Party Solution

5.1 Overview

Following the work of YoGURT, the initial attempt at an in-house solution, and Autodash, the in-house solution using a dashboarding product, a purely third-party solution will also be evaluated for completeness. DeltaDNA is chosen as the third-party service, and is the subject of a case study demonstrating how a developer might use it. It is then evaluated on the comparison framework presented in Chapter 2.

5.2 Introduction

As was discussed in an earlier chapter (Chapter 3), YoGURT, the initial exploration of analytics solutions, has a number of limitations, most notably lack of effective data visualization, which leads to ineffective analysis of the data. Autodash (Chapter 4) sought to address these limitations, largely through improving the automation process, and improving the visualizations through the use of dashboarding.

In-house solutions are frequently effective for many independent studios, with one indie developer specifically stating that analytics solutions allow for usability testing where resources do not allow traditional qualitative methods [30]. However, there is a distinct disadvantage of these tools in that they require time and resources to be
created. There is value then, in considering third-party tools which may also meet a
development team’s requirements. One must understand that the goal of this thesis is
not to discuss only internal solutions available to small-size developers, but is instead
focused on which user research methodologies are most effective for small-size studios,
given smaller resources.

One such third-party tool available to developers to use is DeltaDNA. DeltaDNA
is a popular analytics solution used by many independent game studios, such as Gaijin
Entertainment\textsuperscript{1}, Angry Mob Games\textsuperscript{2}, and Eutechnyx\textsuperscript{3}. This chapter will discuss the
benefits of adopting a third-party tool for analytics use during development, using
DeltaDNA as an example.

\textit{Disclaimer: As much of the content of this chapter was explored on a protected
intellectual property under development, some of the information has been withheld.}

\section{5.3 Motivation}

As was reported in an earlier chapter, third-party solutions can be set up faster than
an in-house solution \cite{62}. A significant portion of this increase in speed can be at-
tributed to not needing development time to get a system in place. Established,
long-standing developers may find that in-house solutions cost less in the long-term,
but the short-term value of a third-party solution cannot be understated. Addi-
tionally, these solutions will generally be tried and tested, are going to be reasonably
effective solutions. A developer will be able to understand exactly what they are
using.

Third-party solutions can generally be divided into \textit{products} and \textit{services} \cite{31}. Products are typically one-time purchases, and are individual systems that can be
added to a game or solution, much like Finalboard was for Autodash. Services are

\begin{footnotesize}
\begin{itemize}
\item \textsuperscript{1}gaijinent.com
\item \textsuperscript{2}angrymobgames.com
\item \textsuperscript{3}eutechnyx.com
\end{itemize}
\end{footnotesize}
generally sold through licenses, which require monthly fees. Services are convenient in that they should be updated and maintained constantly by their developers, and generally require no user installation, with the service provider acting as host [31]. As the previous chapter discusses Autodash used a product, Finalboard, this chapter will be focusing exclusively on third-party solutions as services.

The benefits and drawbacks of third-party solutions vs in-house solutions in general [63, 64] have been discussed at length outside of the field of gaming. The arguments in favour of third-party solutions can be summarized as: (1) they are complete, and likely will have solutions to problems that may not be known yet, and (2) they significantly reduce development time [62]. The negatives can also be summarized: (1) they may come with additional, unneeded bloat, and (2) they may not be specific enough for a developer’s needs [62].

As this thesis has already presented an in-house solution in YoGURT (Chapter 3), and an in-house solution using a third-party product in Autodash (Chapter 4), there is benefit in also showcasing an entirely third-party solution as a service, for completeness. Continuing from the work presented in the previous chapter, the stream game from Chapter 4 will be evaluated with a third-party solution.

5.3.1 Studio Needs

The stream game described in the previous chapter underwent a number of changes, most notably requiring a new server/hosting architecture. The existing database with Autodash became unusable. There was significant time pressure for the development team to get access to a new server, and to re-integrate Autodash. However, this was instead taken as an opportunity to consider third-party solutions for the analytics portion of the stream game.

The development studio of the stream game, as a small-size studio, can act as a representative of small-size studios in general. Thus, as the third-party solutions are
to be evaluated on how effectively they can optimize the user research process for the stream game, one can extrapolate how effective they might be for small-size studios in general.

5.3.2 Requirements

The requirements for a third party tool vary for each individual game and studio. In this particular instance, the developers wanted to be able to replace what had been lost in the server change. The third party solution would need to be a service that could be responsible for as much as possible in the replacement of the old server.

As a service, the solution should be able to provide and host a server and database [31]. Additionally, in order to be effective, the service's database should allow for direct data access, preferably in the form of SQL, or through SQL stored procedures [54]. Understanding that the solution would be used to present information for an internal audience, some manner of dashboarding and visualization would be beneficial as well [50]. As a final requirement, the time to implement this solution must be less than recreating the Autodash setup, otherwise there is no real value in pursuing alternative solutions.

To reiterate, in order for a third-party tool to be effective in this instance, specifically for this studio, it needs to meet these requirements:

1. Allow for the creation and management of a database
2. Allow for the ability to call saved procedures
3. Visualize analytics data (graphs)
4. Show multiple graphs in a dashboard
5. Be faster to set up than Autodash

The final point on this list is arguably the most compelling. From a strictly financial point of view, the cost of adapting existing tools must be compared to the cost and benefits of adopting a third party tool.
There are a number of tools available which could, with some work, fill these requirements. Notable examples of these game analytics solutions are Amplitude\(^4\), DeltaDNA\(^5\), GameAnalytics\(^6\), Localytics\(^7\), and Mixpanel\(^8\). Each solution presents a variety of strengths and weaknesses, but each of them met all of the requirements adequately. However, some solutions met the requirements more effectively than others. For example, DeltaDNA allows raw SQL access to its game databases, as compared with other solutions requiring additional plugins to interface with the database, so analysts experienced with databases may find it a more powerful tool. In fact, developer familiarity with the tool combined with the direct SQL access was the reason that DeltaDNA ended up being preferred over the other solutions.

5.3.3 DeltaDNA

DeltaDNA was chosen as the example third-party tool, to demonstrate the potential benefits of third-party solutions. DeltaDNA is a game analytics and marketing platform used by several game companies. It features a plugin SDK which sends events to its own hosted server, much like YoGURT does. For developers with fewer than 200 000 monthly active users, DeltaDNA costs only $100 per month [65]. As mentioned earlier, one of its strongest, unique features is that it allows analysts direct SQL access, meaning that databases can be created and managed, stored procedures and queries can be called, and data can be processed using DeltaDNA’s own servers. These satisfy the requirements (1) and (2) listed in the requirements section.

DeltaDNA also includes a graphing and dashboarding tool. With direct access to the SQL database, and a graphic widget-builder, DeltaDNA offers a much more accessible dashboard than Autodash, which requires an understanding of coding that may

\(^4\)https://amplitude.com/
\(^5\)https://deltadna.com/
\(^6\)http://www.gameanalytics.com/
\(^7\)https://www.localytics.com/
\(^8\)https://mixpanel.com/
not be reasonably attainable for a non-programming analyst. These visual features satisfy the requirements (3) and (4) presented in the requirements section.

The visualizations and the unique strength of direct SQL access makes DeltaDNA the most compelling solution for the analysts of the stream game. Recreating the SQL database of the stream game would need to be done, either on a hosted server, or done through DeltaDNA’s own database management. This single feature can dramatically reduce the adaption time of the new tool. At its worst, DeltaDNA can be used as a remote server, hosting the game database. At its best, it could reduce dashboard creation time, widget creation time, and maintain fast server access for analysis.

Understanding that integrating Autodash requires an amount of programming knowledge that might not be reasonable, and that DeltaDNA greatly facilitates the creation of dashboards and widgets, it can be seen that this third-party tool would be a more time-effective solution than using the Autodash. From this, the final requirement, (5) is met, and DeltaDNA was chosen to be used for the stream game.

5.3.4 Stream Game

The stream game is the same one presented in Chapter 4, discussed in more detail in section 4.3.2. The focus of this game is the interactions of the spectator-players through the viewing client.

Remembering that active spectator-players are rewarded with currencies, which are used to further engage with the broadcast, it is still extremely important to balance and manage the economy of the game. The amount of currencies being generated for the spectator-players, as well as the main sinks of that currency are all important to assess. In addition, understanding which items users are purchasing is very important to know, as well as the reasoning behind these decisions. At this point in time, the game only allows spectator-players to purchase items at designated “buying phases”,
which boosted the strength of enemies in the broadcast for the next coming enemy wave in the broadcast.

5.4 Usage

DeltaDNA was chosen as the tool to use for the stream game, based on the information presented earlier in this chapter. It was used in a case study to determine how effectively it can be used to optimize user research methods for small-studios. As was the case with Autodash, this section of the chapter is not done to assess DeltaDNA by seeing how it improved the stream game, but instead to demonstrate how a small-size team of developers could use DeltaDNA to facilitate user testing.

5.4.1 Case Study using DeltaDNA

The user test was a session run with 9 participants acting as viewers. There were additional viewers participating, but these were not counted in the user testing session. Each tester was told to watch the broadcast stream, and interact with it however they felt was appropriate. All of their interactions in the viewing client were recorded in the database.

For this user test, the users started with an amount of 8000 currency to spend. The stream game offered the viewers two buying periods in which they could purchase items to increase the strength of the enemy waves. There were four possible purchases that could be made, each item increasing in value from the last.

Building off of the user test ran with Autodash from the previous chapter, and following research into mixed-method [35] user testing, analytics were used as a supplement to a focus group of the 9 viewers. They were asked questions regarding how they interacted with the viewer game, and analytics were used to focus in on specific buyer habits or to highlight potentially interesting pieces of information, which would spur discussion.
5.4.2 Analysis

DeltaDNA was used after the user testing session to quickly produce analytics measures which would help guide discussion in the focus group. However, the servers hosted by DeltaDNA were not updated with the game data for some time after the testing session, so the analytics data was not available until the focus group had already started. While this was known to a certain extent, as DeltaDNA states the information would be available within 5 minutes [66], in this instance the time was found to be 15 minutes. While potentially a one-time error, it is worth mentioning here. However, when the data was present, it was quite useful. Consider the following charts:

First Wave Purchases by Viewer

Figure 5.1 shows the amount of currency spent on different items, organized by the different viewers. This chart was used in the focus group to highlight that there were almost no purchases of the first item. The responses from that comment prompted the viewers to mention that they felt the leftmost item would be the least impactful, as they assumed there was a progression in power form left to right of the items. Had this data and analysis not been present, this perceived progression in power might not have been mentioned.
Figure 5.1: A breakdown of the currency spent by each spectator by the item they purchased, during the first purchasing phase

**Second Wave Purchases by Viewer**

This chart (Figure 5.2) also shows the amount of currency spent on different items, organized by the different viewers, but for the second wave of purchasing. This chart was used in the focus group to question about changes in buying behaviour. For example, the second last spectator-player bought only the fourth slot item in the second wave of purchasing, after having sampled everything in the initial wave. This, when probed in the focus group, highlighted that the spectator-player felt the fourth slot item was the strongest, and was the only item worth purchasing. Thus, they spent all their currency on it. Without this data and analysis present, the spectator-player would not have been probed for additional information, and it might not be known that the spectator-player would buy only what they thought had value.
Figure 5.2: A breakdown of the currency spent by each spectator by the item they purchased, during the second purchasing phase

**Currency Spent / Received by Viewer**

This chart (Figure 5.3) aims to show the amount of currency earned by a playtester for a session and the amount of currency spent during the session. Not pictured in this chart is the 8000 starting currency. From this chart the viewers were asked whether or not they felt they were earning enough currency based on their spending habits.

Figure 5.3: A breakdown of the currency spent and received by each spectator
5.5 Discussion

The charts presented in this chapter are not the ones created by DeltaDNA’s visualization widgets. The main analyst of the stream game chose to not use the dashboarding tools present in DeltaDNA. Instead, the SQL database hosted by DeltaDNA was queried using stored procedures. This information was moved into Excel, where it was processed and manipulated more easily. This was unfortunate, as this was a strong selling point of DeltaDNA.

Another drawback of using DeltaDNA was that the database did not update immediately. While a true real-time dashboard is not feasible outside of internally-hosted servers, it would have allowed for potentially more discussion in the focus group. This is not a problem that would have occurred with the Autodash, which is able to update and query the database as much as is necessary.

Despite these negatives, a successful user test was still performed using DeltaDNA, and the analytics were able to provide useful feedback in conjunction with a focus group. As an analytic solution in mixed-method user research, DeltaDNA was able to optimize the user evaluation process for a game in development, which is a boon for these small-size studios.

5.5.1 Comparison Framework

After having shown a case study of DeltaDNA, it can now be assessed based on the comparison framework presented in Chapter 2. The key points of this analysis are (1) affordability, (2) shareability, (3) data refresh rate, (4) custom data analysis, (5) visualization quality, and (6) customizability.

Affordability: DeltaDNA, being a third-party solution, does have a cost to use. It is a monthly fee of $100 per month for developers with less than 200,000 monthly active users (MAUs). During development time, the only active users a studio need concern themselves with are playtesters, so the amount of MAUs will most definitely
be under the 200 000 threshold. In terms of time to implement, an inexperienced developer could have a reasonably full-featured DeltaDNA presence in their game within two weeks.

*Shareability:* All data housing and data processing is reliant on DeltaDNA’s servers. In order to access the information on the DeltaDNA dashboards, one needs to be logged into an approved user account. These accounts must be managed by an administrator, who needs to invite individual users. This makes the information secure, but does make sharing tedious with non-internal audiences.

*Data Refresh Rate:* DeltaDNA is reliant on its own servers, which are used by multiple studios. Because of this server traffic, the servers cannot process information instantaneously. DeltaDNA states that the game servers are updated and can be queried within 5 minutes [66] of the events being sent. However, in practice from the user tests run in this chapter, this ended up being closer to 15 minutes before an analysis could be made.

*Custom Data Analysis:* DeltaDNA uses a web portal as an interface for the SQL database it hosts. There is no need to write scripts that interpret or format the data, raw access is driven. This is very good for analysts with limited programming knowledge, as none is required to query the database.

*Visualization Quality:* DeltaDNA has a complete graphical chart builder, with direct access to the database. This makes DeltaDNA graph building very accessible to non-programmers. Based on the case study presented in this chapter, the charts can act as a quick analytics report after a user test, which is helpful for small-size developers.

*Customizability:* DeltaDNA offers custom event tracking within its framework, and the creation of custom dashboards for the presentation of data. It does not allow raw HTML and code access but does offer enough flexibility that raw access is not needed.
5.6 Summary (Chapter 5)

As a developer, there are considerations one must have in regards to choosing between different solutions for user research. In-house methods may be used, or third-party tools may be used as well. Following research from the other chapters, DeltaDNA acts as an example of a third-party solution to be evaluated on how effectively it can optimize user research for small-size developers. Based on findings from a case study using DeltaDNA, it is found to be an effective analytics tool in a mixed-method approach to games user research.
Chapter 6

Discussion & Conclusion

6.1 Overview

This chapter will be comparing and contrasting the different kinds of methods that may be used in an affordable user research setting for analytics. As the video game industry is becoming increasingly large, there is value in ensuring quality in the products of the field. However, smaller, independent studios generally do not have the resources to have a dedicated team of GUR researchers. These smaller developers still are in need of GUR solutions to ensure the quality of their product, thereby presenting a need for a low-cost, accessible means of performing formative game evaluation. Analytic methods can be used in a mixed-method approach to performing user research, which can optimize the user research process. This is beneficial for small-size developers that require evaluations for their games.

Chapter 3 briefly discussed an initial foray into analytics solutions, demonstrating an example of an analytics solution built fully in-house, called YoGURT. YoGURT, while extremely limited, shows promise as an analytic solution. With some time investment, small-size game developers can have access to analytic solutions like YoGURT. Even the minimal example of YoGURT has benefits to use, and can be used as a reference guide for developers creating their own in-house analytics solutions.
Chapter 4 took a more in-depth look into in-house analytic solutions, showcasing Autodash, a largely in-house solution using a third-party dashboarding software. It was built with the goal of automating playtest reports. While it did not fully succeed at this goal, it does facilitate the analysis by automating datamining. In this regard, it does help optimize the evaluation process for smaller-size studios.

Chapter 5 discusses DeltaDNA, a third-party analytics service. It acts as a solution to be evaluated on how effectively it can optimize user research for small-size developers. Based on findings from a case study using DeltaDNA, it is found to be an effective analytics tool in a mixed-method approach to games user research.

6.2 Discussion

As was presented in earlier research and in and earlier chapter (Chapter 2), the key points of comparison for these different methodologies are: (1) affordability, (2) shareability, (3) data refresh rate, (4) custom data analysis, (5) visualization quality, and (6) customizability. The three solutions being compared in this thesis are fully in-house solutions like YoGURT, hybrid in-house solutions with third-party products like AutoDash, and third-party solutions that are services, like DeltaDNA.

6.2.1 Affordability

The affordability of a solution refers to both the cost in dollars to adopt a tool, as well as the opportunity cost of creating a tool, by losing production time. This is an extremely important aspect to consider between solutions, as small-size studios typically have very minimal resources.

YoGURT, as a solution developed entirely in-house, required no purchasing and therefore had no acquisition cost to speak of, making it seem like an attractive, affordable option. However, it did have a development time of about a month, on a team with three developers who were largely inexperienced with analytic solutions. The DLL aspect of the tool was done in only about a week, meaning the majority of
this development time was spent on the server-side architecture, and the webhost of the visualization half of the tool. While a more experienced team would be able to create a similar tool in a much smaller timeframe, a team with much less familiarity would cost a lot in terms of tool development.

AutoDash is also largely an in-house solution, meaning that its up front cost was minimal. However, it also used a dashboarding framework, which would be a 50\textit{investment for a single project}, or 500 for multiple projects. On top of this one-time purchase, there is a cost of developing for this dashboard and making widgets work. AutoDash itself was completed by a single developer with limited experience within the span of a month. A developer more experienced with PHP, Javascript, AngularJS, and SQL could develop a similar dashboard in a shorter timeframe, but again, a team with inexperienced developers could end up costing lots in terms of development. However, it is worth mentioning that AutoDash is only a visualization tool, in order to have data recording, additional time would need to be spent on creating a metric recording tool like YoGURT has, adding to the development time.

DeltaDNA, being a third-party solution, has a cost associated with using it. For indie developers, or rather, developers with less than 200 000 monthly active users (MAUs), DeltaDNA charges a monthly cost of $100 to use their services, which is said to be 90\% of their regular cost [65]. During development time, the only active users a studio need concern themselves with are playtesters, so the amount of MAUs will most definitely be under the 200 000 threshold. However, DeltaDNA does not require nearly as much time to adopt, and it can be set up in a significantly shorter time. An inexperienced developer could have a reasonably full-featured DeltaDNA presence in their game within two weeks.
6.2.2 Shareability

Shareability, in this context, refers to how easily the results of an analysis can be distributed to other members of the development team. This is important, as it facilitates discussion between group members, and makes for faster reporting after a user test.

YoGURT, as an in-house solution, requires a host for the SQL database. The web host also has a web page which queries the SQL database, and presents this information to anyone viewing the webpage. In this respect, the information’s shareability is limited only by the server’s ability to process users. As an added benefit, because the hosting must be done in-house, there is the potential that the webpage can be hosted only on a local intranet, ensuring that potentially sensitive user testing data can never be publicly accessed.

AutoDash is also an in-house hosted solution. The web page running the PHP scripts that query the SQL databases is available to anyone with access to the web host. This makes the sharability of AutoDash, and of any other in-house solution, limited only by the server architecture. Again, as with YoGURT, this server could be hosted on a local intranet, kept away from public access.

DeltaDNA is not an in-house solution, and as such all data housing and data processing is reliant on DeltaDNA’s servers. In order to access the information on the DeltaDNA dashboards, one needs to be logged into an approved user account. These accounts can only be managed by an administrator, who needs to invite individual users. There is no convenient way to allow guest access to the information presented in a dashboard. This makes the information more secure, but does make sharing it somewhat more difficult.
6.2.3 Data Refresh Rate

The data refresh rate refers to how quickly a game database can be updated with information after a user test. The need for this varies greatly depending on developer needs. However, to be used in a mixed-method approach to user research, it is important to get the data updated and queried as soon as possible to allow for a more optimized user research experience.

YoGURT, as an in-house solution, is limited only by the speed at which the server may process input. From this, the data can be queried and updated as frequently as needed. In general practice, this should never be less than 1 second.

AutoDash is also limited only by the server hosting it. The particular dashboarding framework used does not limit the refresh rate, so the only bottleneck is, again, the host. This means the data in AutoDash can be up to date within 1 second.

DeltaDNA however, is reliant on its own servers. In order to handle the amount of requests that may be processed on their servers, as there are over 100 million MAUs for each of the games hosted on DeltaDNA [67], the servers cannot process information that quickly. DeltaDNA states that the game servers are updated with event information that can be queried within 5 minutes [66] However, in practice from the user tests run in Chapter 5, this ended up being closer to 15 minutes before an analysis could be made.

6.2.4 Custom Data Analysis

The custom data analysis refers to what data can be accessed, but more specifically, how the data can be accessed. This is in terms of data formats. This is an important consideration for what analytics tool to use, as understanding how the data is stored, or how data needs to be formatted for that tool, can help developers decide which analytics solution to use.
YoGURT relies on tables to present information, as well as Google charts. The tables are created in PHP scripts which pull information directly from the SQL database. The Google charts visualizations require the data to be formatted in JSON, so the PHP scripts format the results of a SQL query.

AutoDash also relies on tables and Google charts, using the exact same data formatting methods as YoGURT. However, these methods are also much more involved than YoGURT, as the amount of code needed to pivot and format the SQL queries into a Google Chart is rather large. This method requires a fair amount of programming knowledge from an analyst.

DeltaDNA does not require any PHP scripting or HTML coding, as the web portal acts as an interface for the SQL database hosted on DeltaDNA. There is no need to write scripts that interpret or format the data, RAW access is driven. This is better for a non-programming analyst, as there is no need to learn coding techniques in order to query the database in a meaningful way.

6.2.5 Visualization Quality

The visualization quality of a tool is largely related to accessibility, and goes hand-in-hand with the shareability of the tool. The more general the audience of the tool will be, the more important it is to have effective visualizations which can summarize the information of the tool.

YoGURT represents data in the form of a table full of the events that occurred in a play session, as well as a bar chart representing the frequency of those events. This is an extremely limited visualization, as it does not allow for further drilling down. Ultimately, it does not help a user test very much because of it’s limitations.

AutoDash was made after the visualization shortcomings of Yogurt, almost as a direct result of them. The visualizations are significantly improved, there are multiple charts on a single page as a result of the dashboarding framework. On top of this,
there are a variety of charts that can be used, most notably bar charts, column charts, and line graphs. Between these visualizations, the dashboard can act as a quick and reliable analytics report after a user test, making it very friendly for independent developers.

DeltaDNA has a complete chart builder built-in, with raw SQL access to populate the charts. The graphic chart builder UI allows for many visualizations to be made, making the dashboards presented in DeltaDNA very approachable for both those preparing the dashboard, and those viewing it. These visualizations also act as a quick analytics report after a user test, which is helpful for the independent developers.

6.2.6 Customizability

Customizability refers to how easily a given tool solution can be adapted to a different project. As an extension to this point, it also refers to how much control a developer has over the tool itself, if they wish to modify it.

YoGURT, as an in-house tool, is completely open source to the developer. Every single aspect of it may be changed and customized as needed. This is one of the strongest aspects of an in-house solution, as long as development resources can be spent, it is infinitely customizeable.

AutoDash is also an in-house tool, however, it does have closed-source components in the form of the dashboarding framework. The dashboarding framework itself was made to be modular to allow for customization, but it is still a closed-source system. Despite this, it is still an extremely customizable solution.

DeltaDNA, being a third-party solution, cannot offer the same amount of customizability as the in-house solutions can. However, it does offer custom event tracking within its framework, and the creation of custom dashboards for the presentation of data. While it does not allow for the raw HTML and code access of an in-house solution, it does allow enough flexibility that this raw code access is not needed.
6.3 Thesis Contributions

The research questions presented at the beginning of this thesis are as follows:

1. How can the cost of GUR be reduced for developers?

2. What analytics solutions are available to small-size studios to perform user research?

3. What recommendations can be made for small-size studios looking to perform user research?

The first of these questions can be answered from research presented in Chapter 2. Following what was presented in Microsoft’s TRUE solution [16], and the work of Canossa to mix qualitative and quantitative methods [68], mixed-method approaches can help optimize the time and cost of performing user research. Even something as simple as the example of the 12 minute lull in a stream game broadcast discussed in Section 4.5.4 of this thesis can stand as a testament to this. Using analytic methods can help find focus areas for other user research methodologies.

The second of these questions is answered throughout the chapters of this thesis, where Chapter 3 discusses pure in-house solutions, Chapter 4 discusses an in-house solution using a third-party product to handle dashboarding, and Chapter 5 discusses completely using a third-party solution. The challenge here is for these small-size developers to understand what solutions are available to them, and which solution better fits their needs.

The final question, on recommendations for the small-size studios looking to perform user research, is difficult to answer. Each small-size developer is going to have different needs for an analytics solution. To help discuss and compare analytics solutions, this thesis presents a framework of categories to aid decisions for small-size studios.
6.3.1 Comparison Framework

This thesis presents 6 categories for consideration when selecting analytics solutions for small-size studio. The categories are (1) affordability, (2) shareability, (3) data refresh rate, (4) custom data analysis, (5) visualization quality, and (6) customizability. The goal of this comparison framework is to highlight the important areas of need that small-size developer have, in order to determine what analytic solution may be best for their game in development.

6.3.2 Overall Comparison and Recommendations

This these presents 3 different analytics solutions: Yogurt, Autodash, and DeltaDNA. Each of these solutions is subject to a breakdown by the comparison framework. been simplified and represented in the following table:
<table>
<thead>
<tr>
<th>Metric</th>
<th>YoGURT</th>
<th>AutoDash</th>
<th>DeltaDNA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Affordability</td>
<td>No purchase</td>
<td>$50 dashboarding framework cost</td>
<td>$100 / month fee</td>
</tr>
<tr>
<td></td>
<td>Up to a month of development time</td>
<td>Up to a month of development time</td>
<td>Up to two weeks of integration time</td>
</tr>
<tr>
<td>Sharability</td>
<td>Available on a hosted webpage</td>
<td>Available on a hosted webpage</td>
<td>Available through the DeltaDNA portal</td>
</tr>
<tr>
<td></td>
<td>Limited only by the server architecture</td>
<td>Limited only by the server architecture</td>
<td>Users need to be added to the DeltaDNA account</td>
</tr>
<tr>
<td>Data Refresh Rate</td>
<td>As low as 1 second</td>
<td>As low as 1 second</td>
<td>5 minutes</td>
</tr>
<tr>
<td>Custom Data Analysis</td>
<td>JSON data from PHP scripts that process SQL queries</td>
<td>JSON data from PHP scripts that process SQL queries</td>
<td>Direct, raw SQL access</td>
</tr>
<tr>
<td>Visualization Quality</td>
<td>Limited visualization only bar chart</td>
<td>Good visualizations Bar, pie, line charts</td>
<td>Good visualizations Bar, pie, line, funnel</td>
</tr>
<tr>
<td></td>
<td>No graph builder</td>
<td>No chart builder tool</td>
<td>Comes with a chart builder tool</td>
</tr>
<tr>
<td>Ability to Customize</td>
<td>In-house solution Source code is available to modify</td>
<td>In-house solution source code is available to modify</td>
<td>Custom Events Custom Charts</td>
</tr>
<tr>
<td></td>
<td>Potentially limitless customizability</td>
<td>Dashboard framework is closed source</td>
<td>Not completely customizable But what is offered is fairly complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Potentially limitless customizability, regardless</td>
<td></td>
</tr>
</tbody>
</table>

### 6.3.3 In-House VS Third-Party

There is no one simple answer for whether or not an indie team should be using an in-house solution or a third-party solution. The most important thing is to understand the needs of the specific independent developer. For instance, an independent developer with absolutely no analytic experience, and limited experience with web development and server architecture would probably be wasting time and resources attempting to build an in-house analytic solution. However, a developer with significant amounts of technical experience in app and tool development might find it much quicker and therefore cheaper to construct an in-house analytic solution.
Additionally, the needs of the games in development must be considered. In the case of a live user test, it might be imperative for developers to have data up to date within 5 seconds of it happening in game. This need for such a fast data refresh rate immediately means that a third-party solution like DeltaDNA cannot be used, as the data refresh rate is simply not fast enough to effectively run a user test.

Another point that may affect a developer’s decision is the state of completion of the game. In very early stages of development, the game is likely to change a lot; having dashboards and visualizations built on complex server code like Autodash is not ideal, as these events tracked may need to change, thus can be a waste of development time. In this regard, a much simpler even tracking solution like YoGURT, or a pre-built solution like DeltaDNA may be more appropriate. In later stages of development, the simple visualizations of YoGURT may not be sufficient, and thus a tool more like Autodash or DeltaDNA would be more appropriate.

Table 6.2: Recommendations based on tools

| Yogurt | Developers with a limited budget, but technical know-how  
Developers who wish to heavily customize their solutions  
Best started at the beginning of development where it can be built alongside their game code |
|--------|-------------------------------------------------------------|
| Autodash | Developers who already may have a database with event tracking  
Analysts who frequently need to make playtest reports  
Developers who are familiar with PHP and Javascript  
Developers who wish to customize their solutions  
Best used after game concept is finalized, and event tracking does not need to change much |
| DeltaDNA | Developers who do not have the technical knowledge to make an in-house solution  
Developers who are under significant time pressure to perform user testing  
Analysts who are very proficient with SQL  
Can be used any time during development |
6.4 Limitations and Future Work

6.4.1 Small-Size Studio Representation

The comparison framework in this thesis has a somewhat obvious limitation in that it is used to compare analytic solutions that are being used on only a single game from a single small-size studio. The needs of the stream game and developer may not necessarily be representative of all small-size studios, and this must be noted as a limitation. However, despite this limitation, the framework could reasonably be expanded into different game genres and across different developers, as it is a framework for assessment made for general use.

6.4.2 Creating an Improved Solution

To argue for what would be interesting and relevant future work, one must reconsider the strengths and weaknesses of each of the methodologies that have been presented in this thesis. Understanding the weaknesses of each of the tools will allow for understanding into how to make a solution which would be able to be used more generally, as an open-source analytics solution that could rival paid services like DeltaDNA, without any of the negatives presented of DeltaDNA.

YoGURT is a tool that is overly simplistic, featuring limited event tracking and a server architecture that is overly complex for what it needs to accomplish. This server architecture makes it difficult to effectively pull data for analysis. However, as a positive it presents an extremely simple to use telemetry gathering DLL, which facilitates event tracking. As another positive, the visualizations follow a “drill-down” design philosophy, making them effective for a broader audience.

Autodash is a tool that allows for more complex visualizations, but requires a telemetry gathering tool. A large critique of Autodash is that while the dashboard automated the analysis, building the charts and scripts required lots of programming
knowledge, and as a result, it was not the easiest tool to use in this regard. However, as a positive, the automated dashboarding was quick, and was able to be used to optimize user experience testing.

DeltaDNA, the third-party tool, does not have many weaknesses outside of one: it is hosted on its own servers, so it must limit the amount of requests at a given amount of time, and it may take a while to update databases as a result. While this could also be seen as a positive, that DeltaDNA hosts itself for a fee, thereby making the solution attractive for those with limited server knowledge, it does not allow for a local installation. A local installation would allow for significantly faster data retrieval and refresh rate. DeltaDNA also has a graphical widget builder, making it very easy to make dashboards.

Given the presentation of the tools above, and understanding the weaknesses of them, future work would find the creation of a combination or hybrid of YoGURT and Autodash, to overcome their weaknesses. Using YoGURT’s telemetry gathering DLL, sending data to an improved server structure, and displaying that information using Autodash’s dashboarding software alone would be an improvement. However, following the benefits of DeltaDNA, there would also be benefits to having a graphical widget builder for the dashboard, so that less programming knowledge is needed to use the tool effectively. It would essentially become an offline, local installation of DeltaDNA. This tool would be made as open source as possible, allowing others to use and customize it for free, as essentially a third-party product which can be fully customized.

The improved solution, once created, would be used with multiple game studios working on a variety of different games. This is done to address, in part, the limitations of this thesis with respect to being representative of small-size developers.
6.4.3 Other Applications

The comparison framework proposed above is not limited only to games. While the basis and motivation for the research was to help small-size developers create GUR solutions to optimize user research for games in development, the framework could easily be applied to non-game products. This is especially relevant today with companies using gamification to make users feel more involved when engaging with tasks [69]. As more things become “game-like”, there is more need for evaluation of these user experiences. Therefore, as potential future work, a case study of a non-game user experience would be interesting, using the comparison framework presented in this thesis.

6.5 Conclusion

As the gaming industry continues to expand, more and more developers will be able to make games for a larger audience, thanks to the growing demographics in players, and the improved access to gaming platforms. Additionally, there is increased accessibility for developers with publicly accessible SDKs for Android and iOS devices, and game engines like Unity3D and Unreal. It is easier now than it ever has been to both make and play games. However, these developers will require their games to be evaluated to ensure quality.

User researchers have been adopting HCI evaluation methods, traditionally used for productivity applications, and applying them to games. This process is known as Games User Research, or GUR. GUR is largely focused on using combinations of qualitative and quantitative measures in order to improve the user experience of games. Independent and small-size studios often find that it is difficult to effectively perform GUR however, as they typically lack the resources to do so. Some researchers propose using a mixed-method approach with analytics to optimize this evaluation, making it more accessible and affordable for the small size studios.
As there are a number of analytic options available for developers, this thesis contributes a framework for comparing different analytic solutions. Additionally, this thesis compares three different solutions: an in-house solution (YoGURT), an in-house solution with a third-party dashboard (Autodash), and an entirely third-party solution (DeltaDNA).

The framework suggests that all three solutions can be effective for optimizing user testing for games in development. However, the different solutions cater to different developer needs. Before using a solution, it is important to understand the needs of the game and the developers. The comparison framework serves to highlight six points of intrigue that a developer should consider before choosing an analytics solution.

To summarize, while small-size studios typically lack the resources to perform user studies on their games in development, they can consider a mixed-method GUR approach using analytics to help optimize evaluation. Before choosing an analytic solution, they should consider their needs, and reference the comparison framework. This will help developers get analytics solutions that can better fit their needs, allowing for optimized user test results. This thesis contributes to the field of GUR by demonstrating a means by which small-size studios who may lack resources can more effectively start considering user research. As evaluation on games generally leads to better games, this benefits the field as a whole.
References


94


