Game Telemetry

*Store, Analyze and Improve UX in Game from Player-Choices*

Simon Baghdo

*Computer Game Programming, bachelor's level*

2017

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
BACHELOR’S THESIS

Game Telemetry

Store, Analyze and Improve UX in Game from Player-Choices

Author: Simon Baghdo

2016-05-03

Bachelor of Science in Engineering Technology Computer Game Programming

Luleå University of Technology Department of Computer Science, Electrical and Space Engineering
Abstract

During this project, the main objective was to store and analyze the user choices through game telemetry, in the game Bloodlines. With the goal to adjust the game for each member personally, for an improved user experience. This was done through a constructed database. By saving metrics of player choices and events such as: Most used weapon, attempts per session, session time periods, amount of deaths and highest rate of death cause.

The results got analyzed with the control group settings in mind. Adjustments made were based on a fundamental foundation. In addition a web application with the functionality to enter and change the settings metrics in real time.
Sammanfattning

Abbreviations and Terms

**Game Metric** – Types of game measurements used with Game Telemetry [1]
**Game Telemetry** – Remote data acquired from e.g. User-behavior [2]
**A/B Testing** – A way of testing different versions of same solution on the general public

**SQL** – Structured Query Language
**MySQL** – Database handler utilizing SQL
**C#** – Object oriented programming language
**ASP .NET** – Web framework for building websites
**@Razor Markup** – Server side markup language based on ASP .NET

**Data Model** – A visualization of the database
**Apache** – Web server
**XAMPP** – Web server solution stack package
**MyPhpAdmin** – Tool intended to administrate e.g. MySQL through web browser

**UX** – User Experience
**Post-Production** – Last period of production before release
**IP** – Intellectual Property
**QA** – Quality Assurance

**Unity** – A Game Engine
**Bloodlines** – The project game used as a platform for this thesis in Unity
**UI** – User Interface

**Component** – A script in Unity applied to an object to activate feature to that entity.
**Scene** – A form of loaded environment in unity necessary for separating levels such as main menu & play-mode.
Limitations

The limitations for this project were mostly based on either being too complicated, requiring more time to accomplish with little to no effect or would not reach a final state for presentation:

- **Network Security** – Definitely important when it came to configuring cloud solutions for games run on multiple devices. Although with the current state and for the purpose of the thesis, it would definitely not be needed, because the game would only be run on known devices and the database would be locally configured.

- **Attribute Quantity** – The different elements acquirable from the player were only relevant if it was possible to utilize it effectively, otherwise it was a waste of time and space for this thesis.

- **Stamina** – While the functionality for “Stamina” attribute (an element implemented in the game to manage the cost of attacks), was available in the game. The attribute was unfortunately not implemented properly enough to be an efficient asset for the tests.

- **Shield** – The “Shield” weapon, used by the player. Was not added to the adjustable elements of the thesis. This is due the shield component lacking relevant attributes to adjust.

- **Network Address** – The solution would run on a local network due to the network security limitation and due to the fact that the thesis was simply research based.

- **Aesthetic Front-End** – The web application did not need to look appealing. It was only meant to provide an easier alternative to access the player’s data and modify it.
## Contents

1  Introduction ........................................................................................................ 1
   1.1  Goal and Purposes ......................................................................................... 2
   1.2  Background .................................................................................................. 2
       1.2.1  Current State ......................................................................................... 3
       1.2.2  Local Database ..................................................................................... 4
   1.3  Unity .............................................................................................................. 4
   1.4  Social, Ethical, and Environmental Considerations .................................... 5

2  Method ................................................................................................................. 6
   2.1  Monitoring ..................................................................................................... 6
   2.2  Game balancing ............................................................................................. 7
   2.3  Database ....................................................................................................... 7
       2.3.1  MySQL .................................................................................................. 7
       2.3.2  Apache Server ....................................................................................... 7
       2.3.3  XAMPP ................................................................................................ 7
       2.3.4  MyPhpAdmin ......................................................................................... 8
   2.4  A/B Testing .................................................................................................... 8

3  Design & Implementation ....................................................................................... 9
   3.1  Constructing the Database ............................................................................ 9
       3.1.1  Design .................................................................................................. 10
       3.1.2  Important Indexes ............................................................................... 10
       3.1.3  Tables .................................................................................................. 11
   3.2  Game Set-Up ................................................................................................. 11
       3.2.1  MySqlLoginManager.cs ...................................................................... 13
       3.2.2  LocalData.cs ....................................................................................... 13
       3.2.3  MySqlEntityStatsManager.cs .............................................................. 13
       3.2.4  MySqlWeaponStatsManager.cs ............................................................ 14
       3.2.5  MySqlManager.cs ............................................................................... 14
       3.2.6  Algorithm for analysis ........................................................................ 15
   3.3  Web Application Set-Up ............................................................................... 16
       3.3.1  Front End ................................................................................................ 16
       3.3.2  Back End ................................................................................................ 16
1 Introduction

When it comes to designing good games, a general rule for a game designer is to choose and focus on a selected target audience [3]. After the game has been created and published for that audience, it is also an important rule to keep that audience's interest. The question lies in what keeps a player playing the game. What keeps the players interest? In today’s industry of mobile game production, a commonality easily noticed that solves these problems are solutions such as in-app purchases, daily unlocked objectives and much more, but are these solutions enough to keep the player returning to the game or extends the sessions time from the general 7.55 mins. [4]

By examining the top ranked games on mobile devices. Candy Crush Saga by King, was a great inspiration to this thesis. The game focuses on single player entertainment, by increasing the difficulty for each traversed level. Meaning the game lets you off easily in the beginning to let you understand the game mechanics, but later on challenges the player with more suiting difficulty. This idea kept the players interested by challenging their skills on the game. But what if game designers don't want to rely on levels for a tool to increase the difficulty, but still be able to adjust the game’s difficulty? There was many solutions to this problem, this thesis would handle one of these. In this thesis, through the use of game telemetry.

By storing changeable metrics in a database with players’ in-game data. One could analyze the player’s skill, adjust and balance the game. For a more personal environment. Tailoring the game in runtime for the player occupying it. The choice of using game telemetry, was because it balances the game’s mechanics from player’s skill.
1.1  Goal and Purposes
The goal of this thesis was to modify the single player game Bloodlines, with the functionality of game telemetry and with the purpose of adjusting the game’s difficulty, based on the player’s choices, leading to a personal experience for each player individually. This was done through a sequence of A/B testing of users, to analyze the resulting balance and differences in players’ choices. To accomplish this, three implementations were required to be added:

- A database with tables describing the in-game data, session data, the player’s history data and settings for the control groups, with the feature of A/B Testing.
- A web application (Game Server) connected to the database that follows a custom made protocol of dialogue with the database to easily store modify/add data to it, meanwhile adjust each person’s game through a fundamentally test based analyzation of the current data and receive a new response from the player for coincidentally further analyzation.
- Implement a feature in the single player Bloodlines game, capable of connecting to a database with the help of a custom protocol. The implementation needed to be capable of sending and receiving MySQL query requests about the players’ current session to the database.

1.2  Background
Bloodlines is a rouge-like/dungeon crawler/RPG PC game made by a group students in Luleå University of Technology during a school project 2015. During the thesis, this platform was in production state.

The game had in the current state no goal, it simply randomized new rooms with a new set of enemies each time the player finished a room or reached the next level.
1.2.1 Current State

The game’s current state included features such as random level generation loading six rooms, creep and boss mob, a weapon system where the player could use two weapons simultaneously.

The available weapons for the player was a sword and blowgun (representing short and long range use), with which the player could chose with two different buttons, activating either the left or right hand of the character. If the player hit the left button the character would interact with its left hand and right button vice versa.

Furthermore the Creep and boss mobs in the game also had weapons of choice, the creep with its creepclaw and the boss with its bossfist, fireball, pillar of flame and jump special. All entities are based on health/stamina, when the health of an entity reached zero, it obviously died. When the stamina was lower than the weapons stamina cost, it wouldn’t be able to execute that command. All of these weapons had changeable stats.

Health is shown for the player as a red UI bar, and for the mobs as a red circle beneath the unit. The stamina was shown only for the player as a green bar on the UI (see Figure 1.1). The player could also heal itself for a limited amount of times for each session.
1.2.2 Local Database

The local database was in the current state, simply based on “hard coded” attributes, these are attributes installed on each object based on the type of object, e.g. player-, creep- and boss max health. (Same max health, different values). The player object’s max health and max stamina default value was set to 100 (see figure 1.2).

1.3 Unity

The main tool for the game Bloodlines, was the game engine Unity. The engine worked with the programming languages JavaScript and C#. The previously chosen programming language and the one that would be used for this thesis was C#.

The API used in Unity would mainly be MonoBehaviour. An API well made for multiplatform functionality and easy to implement. The standard methods that would be utilized from the components in this project were the Start(), Update() and OnApplicationQuit() methods.

- The Start() method was called once the object that the script was activated on was first initialized. This method would therefore only be called once each time the object was instantiated (created into the scene).
- The Update() method would be called if the object was already instantiated. It would be called once each frame, until the object got destroyed somehow.
- The OnApplicationQuit() method was called once the application was turned off.
1.4 Social, Ethical, and Environmental Considerations
The solutions: client game, web application and database was all running on a local network. Though the data transmitted between each was not encrypted, but the lack of relevant private information made the application safe for testing participants, if and only if the testing participants were not afraid if their applied nickname was available for a third party.

However, the network security choices are only of non-ethical value as long as the information available is not information sensitive to the tester. Furthermore during the thesis, each participant had been informed of the goal of the thesis and the lack of security that surrounded it, and only after being informed could the tester accept these terms by registering their nickname to the application.

Due to the application being connected to a local network, it was not the best in an environmental perspective, due to the fact that each test required the user to play the game on the same system as the server and database.
2 Method

To overcome the task was to first figure out the requirements: What where the internal and external implementations needed for a satisfying result, internal being the game product that already existed, external being the database and game server.

The thing that was noticeable about the solution, it required implementations in multiple areas:

- Client side configuration for database queries, local in-game storage for holding the relevant metrics.
- Database for storing the metrics.
- Web Application for managing entities in the database.

There was no limit to the amount of data available to be analyzed, the more modifiable in-game attributes, the more variation.

2.1 Monitoring

Monitoring games has been in the industry for many years. But only recently, when the internet got popular has it clearly bloomed to a more financially efficient feature, to a company’s product.

The proto-analytics of arcade machines were not improved in a technical way until the advent of the modern internet some 20 years later. However, this doesn’t mean that there was no monitoring going on.

Looking to the experience of classical software development, extensive testing was brought to bear on the game development process. Instead of algorithms and databases, clipboards and questionnaires were used to monitor the progression and actions of test players. While a big part of the testing process was technical in nature, isolating and fixing bugs in code, savvy game developers also charted the interests of players to better cater to them. [5]
2.2 Game balancing

Balance in games is important for designing the game’s mechanics to give the right expectation, from the story of the game.

*Gameplay was all about making choices and in a poorly-balanced game, many of the choices available to the player was essentially rendered useless. And this, in a nutshell, was why game balance was so important.* [6]

For instance if the game’s story describe a harsh environment. Containing a story of loneliness, unfairness and horror such as the Dark Souls franchise by From Software, the balance of game mechanics, between the player’s capability, and its obstacles, do portray this phenomenon. In the case of Dark Souls, known for its punishing difficulty and unfairness, single attacks from enemies, could lead to the game over screen. But this is a valid game balance due to the story and game design.

From this one can conclude, that much of the choices of a games balance is based on the game design and its chosen target audience.

2.3 Database

The database was configured and built upon four implementations, namely: MySQL, Apache server, XAMPP and MyPhpAdmin. Each filling an individual purpose, each aiding the other for configuration and functionality.

Although there was a number of other available solutions that could be used instead, which may as well have been better, these were chosen because they were easy to configured and worked perfectly together, and of course because of the free license.

2.3.1 MySQL

MySQL was an important element to the project, because it required to utilize a database for storing relevant game metrics. The database management system MySQL was chosen because: it was free, easy to install and beginner friendly. MySQL is a strong candidate among other systems. Additionally it was the chosen system because of the compatibility with the external solutions such as XAMPP and MyPhpAdmin.

2.3.2 Apache Server

The Apache server is a helpful solution to setup a HTTP based server either on a remote address or, in this case local. Also it was easy to configure through the use of XAMPP.

2.3.3 XAMPP

A solution working as an adapter for setting up the Apache server and the MySQL, the XAMPP application works perfectly for managing tedious individual processes into easy steps. From here on the application lets us access the MyPhpAdmin database tool. Although there exists multiple alternatives, the choice of XAMPP was because of it’s easy to use setup.
2.3.4 MyPhpAdmin
A web based database manager, letting its user modify databases located on the Apache server. With the feature of GUI control of tables and their relationships, of course including the SQL direct compiler for modifying tables and its contents through SQL commands, (used in this concept primarily for testing).

2.4 A/B Testing
Also known as Split Testing, is a method used for comparing different versions of a solution e.g. web page or application for the purpose of deciding the better version [7]. This method follows a certain test protocol, based on multiple versions of game settings that each participant is tested with. With the result of each player, being analyzed in mind of which control group the participant is assigned to.
3 Design & Implementation

Figure 3.1. Shows the chosen design for the solution, with the database at the top, the web application to the left and the game to the right. Each with arrows implying the relation between each node.

Above is a visual representation of the flow design for this thesis (see figure 3.1). The picture points out that both the game and the web application, both have a setup protocol, based on query sequences with the database. It was required that both parts both send and retrieve data from the database in real-time prior to update the game while the game was being played.

3.1 Constructing the Database
The database was built, upon tables of relevant in game entities, sessions and histories. The entity table holds data for the units in the game i.e. player, creep and boss. The session table explains detailed information about all the player’s sessions. The history table records these and calculates new general information.

All of these tables are based on the row, “GamerID”. This lets us individualize each gamer data and makes it possible for us to retrieve and modify the game prior to the player.
3.1.1 Design

Figure 3.2. A Data model describing the relationship between the tables in the database

The above appendix (see figure 3.2) is called a “Data model”. It represents each table and its corresponding content. Furthermore it also visualizes the relationship between the different tables through the drawn arrows.

To understand the relationship: Each flailed end part of the arrows, means that something was taken from that table. The other end means that, that particular data is used in the other table.

3.1.2 Important Indexes

The design includes very important indexes for managing the relationship between the tables. First of is the “GamerID” column found in the “Entities” table. The GamerID is an index working as the players account ID. When a player starts the game it would send the submitted nickname to the database and check if it already existed. Whatever the outcome was, resulted to a GamerID integer. This was later on used in all the other tables to identify the user.

Next up is the “SessionID” found in the “Session” table. This integer was important to identify the sessions ID (from the game starting and for each game over). Meaning that each time the player died, it would insert a new session with the corresponding metrics related.

Every GamerID would have a SessionID attached to it, allowing modification in real-time once the player either restarted the game or retried from death.
3.1.3 Tables
- The Session table stored data about how the player fared during the last game it played, from start to a game over. It stored the weapon usage time, the amount of heals used, the amount of levels traversed, the amount of creeps and bosses killed, etc.

- The History table was a way to store the outcome of each and every session for each player. The amount of games played, how many times the player died, the total time played and which enemy it died the most from etc.

- Creep, Player and Boss tables all have common attributes, their Max health and stamina, and their weapon physical damage and stamina cost. (Stamina was not used in this context, but is included to show possibilities of improved functionality).

- The two settings tables: settingseven and settingsodd, are the two constructed versions for the control groups, needed for the A/B testing part of this thesis. These two included limits for different attributes. The limits were the factors used in conjunction with the Session and History tables to conclude new attribute values for the Creep, Player and Boss tables. The “even” and “odd” part of these tables describe if the GamerID that the player has been assigned to was even or odd, would determine the player control group.

3.2 Game Set-Up
The game setup was based on 5 different components: A MySQL manager, a login manager, a local data container (for storing all relevant game attributes), an entity stats manager and a weapon stats manager. What most of these C# scripts have in common was that all of them was implemented with Unity, meaning that the standards from “1.3 Unity” where present.
Figure 3.3 Class Relationship “UML Diagram”

The above “UML Diagram”, depicts the final design, the hierarchy of the components implemented to the game. The arrows point out the relation the class has to the others. As seen, the MySqlLoginManager was highest in the hierarchy. It started the MySqlManager, which in turn started the Local Data. The MySqlEntityStatsManager and MySqlWeaponStatsManager classes started when entities was initialized.
3.2.1 MySqlLoginManager.cs

![Login Screen](image.png)

*Figure 3.4. The login screen once the game starts letting the player know that it needs to sign in with a chosen nickname.*

The Login manager was the starting component. The player was required to submit a nickname to register to the database with, as a GamerID. Once the nickname had been submitted, the manager would start the game’s managers (including the MySQL Manager). The manager would find out if the assigned gamerID was of even or odd value and assign it a control group. The game would take off from there as usual (see in figure 3.3).

3.2.2 LocalData.cs

Is a container for storing local temporary data of the session. This data was private and could only be reached through encapsulated fields of get/set methods. This design was important for security reasons and makes it only possible to reach from the related MySQL Manager. For information about the local data, see the Appendix “9.2 LocalData Variables”.

The `setData()` method would take in arrays that the MySQL Manager had constructed from queries from the database.

3.2.3 MySqlEntityStatsManager.cs

Entity Stats manager was a component attached to every unit in the game, e.g. the player, creep and boss. This would make it possible to overwrite and set the attributes relevant to that object, such as MaxHealth and MaxStamina.

Once the object had been created, the component would set the relevant attributes through the encapsulated methods of the MySQL Manager, which in its case got the result from the Local Data.
Each time the object dies would send a message to the MySQL Manager to confirm that it was dead and let the MySQL Manager handle the rest depending on the object type, e.g. Player, creep or boss, before the object was destroyed.

3.2.4 MySqlWeaponStatsManager.cs
The Weapon Stats manager works exactly like the Entity Stats manager, though with a twist. The component was instead attached to the weapon objects, e.g. sword, blowgun, creepclaw, bossfist etc. Which would make it possible to change the units attack attributes.

Important to add is also that the weapon usage time was calculated for each session while the player had equipped the corresponded weapon. This data, like the Entity Stats manager sent its acquired attributes to the MySQL Manager which stored it to the Local Data.

3.2.5 MySqlManager.cs
This manager is a component attached to the System object of the game which is only initialized once the game start. It works as a general manager, a checkpoint between the database and the game. Once the game starts or when a player revives from a game over, a connection to the database would be configured if not already. Later a check for the players nickname would be sent to the database's Entities table if it exists and get the GamerID back.

Upon game over, the manager would send out the allocated local data of the played session, as a row to the database's Session table. Furthermore the manager has also available methods for the Entity and Stats Managers.

There were three main methods of interest in this manager: UpdateLocalData(), ModifySettings(), UpdateHistory(), QueryInsert() and QuerySelect().

The QuerySelect() and QueryInsert() methods were used to communicate with the database through the MySQL.Data.dll library extension. Both of these methods sent a query to the database server with appended instructions to what the database had to do. Each time these methods were called, the methods first set up a new MySqlCommand with the appended MySqlConnection and the query itself. Afterwards a MySqlDataReader was executed to read out the query to the database. Afterwards the instructions varied depending on the method.

The method QuerySelect() was designed to only retrieve stored data from the database, based on the instructions as a string value. The result could contain multiple values, which the same method was ready for with a coma (",") as the denominator.
QueryInsert() was designed to insert the databases’ tables with new rows or update existing rows. The difference (among others) between the two, was that the QueryInsert() method didn’t wait for a response from the database.

UpdateLocalData() retrieved the current data from the databases’ tables, through a series of QuerySelect()s, and ran the LocalData script’s setData() method, see “3.2.2 Localdata.cs” for detail about the setData() method.

UpdateHistory() was meant to update the history table for the player, it checks if the row for the current gamerID already exists and if so updates the row instead of adding a new one.

The ModifySettings() method was the bread and butter of this whole system. Containing instructions which set the game’s settings in motion. Here, the method would handle the analysis and fundamental choices regarding the player’s playstyle. The method started off by retrieving data from the database to receive the latest version from the database.

3.2.6 Algorithm for analysis
During the analysis, multiple calculations of the data were made, although only when the settings MaxGameCountLimit and MaxTraversedLevelsCountLimit was reached. If so, each and every resulting new variable was checked with the control group settings-even/odd table, to not let it exceed or reach below the set Max/Min values in the settings tables. The variables in bold are the results from the calculations:

\[
\text{CreepMaxHealth} = \text{CurrentCreepMaxHealth} + \text{CreepMaxHealthIncrease} \times \frac{\text{AverageCreepKillCount}}{2}
\]

\[
\text{BossMaxHealth} = \text{CurrentBossMaxHealth} + \text{BossMaxHealthIncrease} \times \frac{\text{AverageBossKillCount}}{2}
\]

\[
\text{DeltaUsageCount} = \frac{\text{AvgSwordUsageCount}}{\text{AvgGunUsageCount}}
\]

\[
\text{PlayerSwordDamage/PlayerGunDamage} \text{ (Depending on which one was used more)}
\]

\[
= \frac{\text{oldSwordDamage} - \text{oldGunDamage}}{\text{oldGunDamage} - \text{oldSwordDamage}} \times \frac{1}{\text{1 + DeltaUsageCount}}
\]

\[
\text{AverageHealsPerSession} = \frac{\text{HealCount}}{\text{GameCount}}
\]

\[
\text{AverageKillCount} = \text{AverageCreepKillCount} + \text{AverageBossKillCount}
\]

\[
\text{AverageHealUsagePerKill} = \frac{\text{AverageHealsPerSession}}{\text{AverageKillCount}}
\]

\[
\text{PlayerMaxHealth} = \text{AverageHealUsagePerKill} \times \text{PlayerMaxHealthLimit} \times \text{GameCount}
\]

\[
\text{CreepDamage} = \text{oldCreepDamage} + \text{AverageCreepKillCount}
\]

\[
\text{BossDamage} = \text{oldBossDamage} + \text{AverageBossKillCount}
\]

The resulting bold values was sent to the database to store under the related GamerID. See “3.1.1 Design Figure 3.2” for variable location in the database and Appendix “9.1 Database Variables” for a detailed variable description.
3.3  Web Application Set-Up

The web application was built in C# where the Front End and Back End were implemented separately. The Front End utilized the framework Bootstrap for responsive and dynamic content and web pages. The backend part of the web page was made with the help of the server side markup language ASP .NET @Razor, for easier implementation of dynamic web content.

3.3.1  Front End

The visual part of the web page was not mandatory to have any aesthetic elements, it would simply be made for functionality. Apart from the previous statement, the reason for the visualization was to simplify the process of displaying and being able to modify the database. Tables were displayed. Like a live-feed of the game for all active players simultaneously.

3.3.2  Back End

Being connected to the front end, includes the requirement to handle the features of the front end. The back end part was implemented to connect to the database, once each time the web page was refreshed. It queries the current state of the tables once receiving a valid nickname, login as.
4 Testing

The testing part was based on A/B Testing, (creating multiple versions for multiple users and analyzing the response in the form of new game metric data from the players). The testing method was based on participants playing the game locally with assigned control group. Based on their given GamerID. For each time changing the factors of multiple attributes relevant to the analysis of the game, so that the game changes per player, for a more varied response.

The modifications made on the relevant in-game factors, was set through default values in the database.

4.1 Preparation

The test required from a user was to first register a nickname on the server-side. On the game side, it would register with the same nickname as the previous, and this process would let the game link the currently GamerID to the monitoring and administration of the same id on the server side. After that the user was simply required to play the game in multiple sessions.

4.2 User Tests

There would not be any kind of survey needed nor an application to answer to. This was because this thesis follows the premise of game telemetry, making it only depend on communication from the game to the database.

The users would be told to play the game as usual from the start and as long as they was capable of surviving. Afterwards when data samples has been acquired, changes would be made without the players notice. A new game would begin for the player with updated game settings. From this the ultimate goal was to get the best game life time for the player with the highest possible difficulty.
5 Result

The beneath results provide information about the sessions for each and every player through their unique GamerID. With graphs created from the data stored in the database.

5.1 User Test

During a period of several days, user tests where set up in the local university called Luleå University of Technology outside the studios of computer game developers and computer artists. Students participating ranged from the ages 20-30. These students where asked to participate in the test for this thesis.

Each participant at a time played the game by first accepting the terms and learning the controls before they signed in and started the game stretching for several retries with recorded sessions each time for not over 5 minutes per participant.

5.1.1 Entities

<table>
<thead>
<tr>
<th>GamerID</th>
<th>NickName</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Simon Bagho</td>
</tr>
<tr>
<td>3</td>
<td>Kyparn</td>
</tr>
<tr>
<td>4</td>
<td>Aggro</td>
</tr>
<tr>
<td>5</td>
<td>Nicklas Larsson</td>
</tr>
<tr>
<td>6</td>
<td>Bygg</td>
</tr>
<tr>
<td>9</td>
<td>Matte</td>
</tr>
<tr>
<td>10</td>
<td>l33t</td>
</tr>
</tbody>
</table>

*Figure 5.1. Above figure was captured from the entities table depicting the stored participants chosen nicknames with related GamerID*

5.1.2 Settings

The values of GamerIDs that was of even types was assigned to the “settingseven” and those of odd value was assigned to the “settingsodd” control group and table.
Figure 5.2. “Settingseven”. This table belongs to the control group for default values.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Type</th>
<th>Kollektioning</th>
<th>Attribut</th>
<th>nullÅge</th>
<th>Standardvärde</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GameID</td>
<td>int(11)</td>
<td>Nej</td>
<td>Inget</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MaxPlayerHealth</td>
<td>int(11)</td>
<td>Nej</td>
<td>120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MinPlayerHealth</td>
<td>int(11)</td>
<td>Nej</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MaxCreeperHealth</td>
<td>int(11)</td>
<td>Nej</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>MinCreeperHealth</td>
<td>int(11)</td>
<td>Nej</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>MaxBossHealth</td>
<td>int(11)</td>
<td>Nej</td>
<td>220</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>MinBossHealth</td>
<td>int(11)</td>
<td>Nej</td>
<td>180</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>MaxSwordDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>MinSwordDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>MaxGunDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>MinGunDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>MaxCreeperDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>MinCreeperDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>MaxBossFistDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>MinBossFistDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>MaxBossFireDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>MinBossFireDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>MaxBossPillarDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>MinBossPillarDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>MaxJumpDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>MinJumpDamage</td>
<td>int(11)</td>
<td>Nej</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>AverageHealCountLimit</td>
<td>int(11)</td>
<td>Nej</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>MaxGameCountLimit</td>
<td>int(11)</td>
<td>Nej</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>MaxTraversedLevelCountLimit</td>
<td>int(11)</td>
<td>Nej</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>AverageCreeperKillCountLimit</td>
<td>int(11)</td>
<td>Nej</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>AverageBossKillCountLimit</td>
<td>int(11)</td>
<td>Nej</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>MaxCreeperHealthIncrease</td>
<td>int(11)</td>
<td>Nej</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.3. “Settingsodd”. This table belongs to the control group made as a test, to try out values with higher effect to the result.
There were similarities in the two control groups (see figures 5.2 and 5.3), such as the limits of “MaxGameCountLimit” and “MaxTraversedLevelCountLimit”. These two limits determined the required effort the player needed to go through before the system started to take effect. The “MaxGameCountLimit” for example set the required games played for the player. While the “MaxTraversedLevelCountLimit”, set the required amount of levels reached. Both of these was set to 1 due to convenience sake so that the tests takes effect faster but though to the cost of quality.

Additionally there was no need for individual settings from each participant because the only difference was the two control groups.

### 5.1.3 Sessions

<table>
<thead>
<tr>
<th>SessionID</th>
<th>GamerID</th>
<th>LifeTime</th>
<th>ScoreCount</th>
<th>HeatCount</th>
<th>HealthValue</th>
<th>MaxGameCountLimit</th>
<th>MaxTraversedLevelCountLimit</th>
<th>CorpSpillCount</th>
<th>RaccoonHitCount</th>
<th>CorpHealth</th>
<th>RaccoonHealth</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>27,827</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 1:10:34 AM</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>54,020</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 1:21:25 PM</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>90,742</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 1:21:44 PM</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>126,190</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 1:22:19 PM</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>227,739</td>
<td>170</td>
<td>32</td>
<td>14</td>
<td>1</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 1:22:49 PM</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>270,201</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 1:24:31 PM</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>410,710</td>
<td>191</td>
<td>144</td>
<td>31</td>
<td>2</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 1:33:08 PM</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>966,111</td>
<td>22</td>
<td>16</td>
<td>0</td>
<td>2</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 2:14:43 PM</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>513,358</td>
<td>142</td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 3:17:12 PM</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>1051,21</td>
<td>218</td>
<td>168</td>
<td>13</td>
<td>4</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 5:24:13 PM</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>1407,38</td>
<td>284</td>
<td>51</td>
<td>15</td>
<td>6</td>
<td>51</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 3:30:32 PM</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>1899,64</td>
<td>563</td>
<td>10</td>
<td>7</td>
<td>12</td>
<td>65</td>
<td>1</td>
<td>40</td>
<td>200</td>
<td>0</td>
<td>5/3/2012 3:38:23 PM</td>
</tr>
<tr>
<td>13</td>
<td>5</td>
<td>121,052</td>
<td>120</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>80</td>
<td>0</td>
<td>5/3/2012 3:52:43 PM</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>346,351</td>
<td>142</td>
<td>120</td>
<td>18</td>
<td>0</td>
<td>35</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 4:06:39 PM</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>544,441</td>
<td>96</td>
<td>58</td>
<td>10</td>
<td>2</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 4:41:47 PM</td>
</tr>
<tr>
<td>16</td>
<td>6</td>
<td>726,086</td>
<td>195</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
<td>5/3/2012 6:15:00 PM</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>90,3421</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 1:06:40 AM</td>
</tr>
<tr>
<td>18</td>
<td>8</td>
<td>129,204</td>
<td>11</td>
<td>106</td>
<td>0</td>
<td>1</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 7:06:53 AM</td>
</tr>
<tr>
<td>19</td>
<td>8</td>
<td>307,040</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5/3/2012 7:12:58 AM</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>206,448</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>35</td>
<td>1</td>
<td>40</td>
<td>220</td>
<td>0</td>
<td>5/3/2012 7:22:50 AM</td>
</tr>
</tbody>
</table>

**Figure 5.4. Session table depicting each session for each GamerID taken from the primary key from the table Entities (See figure 5.1)**

Here we can see each and every session played during the testing period by our participating entities (see figure 5.1). The session section came in handy when analyzing the results in a more detailed perspective.
5.1.4 History

Table 5.5. History table, showing the general and total data from the sessions combined for each GamerID

5.1.5 Player

Table 5.6 Player table, showing each participant's latest player entity data

5.1.6 Creep

Table 5.7 Creep table, showing each participant's latest creep entity data

5.1.7 Boss

Table 5.8 Boss table, showing each participant's latest boss entity data
As seen from the above appended figures 5.1 – 5.8 the data stored was the result from monitoring 6 participant’s reactions to the applied control group (not counting myself). A commonality of the participants, was that all had never played the game before. Each had to learn the controls and had to get used to the game’s mechanics. But all of them survived the tutorial map on the first try either by killing each creep or running past them. This conclusion was reached by looking at the HighestTraversedLevelsCount from the History table of each player.

Additionally the players: “Kyparn”, “Nicklas Larsson” and “Matte” with GamerID: 3, 5 and 9. Had not reached the AverageCreepKillCountLimit limit of 25. The cause of this commonality was because they all belonged to the odd control group. Upon further investigation, these three participants also belonged to the least lifetime with the exception of “l33t” GamerID: 10, who played on the even control group. This led to the conclusion that the odd control group was a more difficult and unbalanced control group than the even control group.

If we now instead inspect two players from the odd control group, namely “Kyparn” and “Matte” GamerID: 3 & 9. Both had very similar aspects in their game history. Both within approximately 500 seconds GameTime and neither had reached the AverageCreepKillCountLimit. The only difference was that the participant “Matte” had reached one level higher than “Kyparn”, level 3. The outcome of this divergence was noticeable in the Player table. “Kyparn” had 206 MaxHealth while “Matte” only had 100 and “Matte” with the BlowGunDamage of 19 while “Kyparn” only 10.

This result could be illustrated if we go back to the section “3.2.5 Algorithm for Analysis within 3 Design And Implementation” chapter. Previously described two relevant algorithms regarding the player damage and player max health.

SessionID 1, GamerID 3 MaxHealth = 100 (Starting value of MaxHealth)
SessionID 2, GamerID 3 MaxHealth = 140

\[
\text{MaxPlayerHealth} = \left(\frac{\text{HealCount}}{\text{GameCount}}\right) / \left(\frac{\text{AverageCreepKillCount} + \text{AverageBossKillCount}}{\text{PlayerMaxHealthLimit} \times \text{GameCount}}\right) \Rightarrow 140
\]

Next lets calculate the PlayerGunDamage = 19 for “Matte”, with the GamerID 9. The algorithm for the new damage required to be complicated because it balances the damage between the sword and the gun.

Knowing that the sword was used more than the gun through the session table:

\[
\text{DamageChange} = \left(\frac{(\text{TotalSwordUsage}/\text{GameCount})/(\text{TotalGunDamage}/\text{GameCount}) \times \text{oldGunDamage} - \text{oldSwordDamage}}{1 + ((\text{TotalSwordUsage}/\text{GameCount})/(\text{TotalGunDamage}/\text{GameCount}))}\right) = 6.468
\]

\[
\text{PlayerGunDamage} = \text{oldGunDamage}(12) + \text{DamageChange}(6.468) \Rightarrow 19
\]
5.2 Result Summary

From the analysis of the metrics acquired from the 6 participants. Provided the conclusion that the odd control group increased the player’s damage and max health, but started off with a higher difficulty. Which led to a shorter session time compared to the even control group.

Though this conclusion was based on the acquired statistics, the lack of quantity and quality QA for this thesis led to poor adjustments.

To summarize, the test sequence was not of the best sort, due to the game platform which had been in an unfinished state. With existing bugs that for instance happened each time a player gave the game multiple inputs simultaneously in the form of attacks. This activated a glitch which led to the character being stuck in an attack animation loop. That in each frame sent messages to colliding objects that they have been hit and coincidentally took damage. This glitch was in some sessions heavily utilized, which removed the important difficulty change, and testing scenarios in the game and made it impossible to balance the game based on the player data. Unfortunately this glitch was unavoidable due to the lack of production time.

Though this only meant that the chosen game platform was not ready for features of monitoring and game telemetry to be efficient enough, the thesis still proved in some points that from the small amount of relevant metrics, game telemetry could still investigate player choices and provide changes from them.

Additionally the chosen control group settings was a bit off in the aspect of game balance. The players from the odd control group showed signs of players reacting differently to the new settings which made them adapt to the new environment that the game had created from the last play through, creating a more dynamic and diverse UX.
6 Analysis

The above graph (see figure 6.1), represents the comparison between the values in the tables Settingseven and Settingsodd. These two tables represent the available control groups for the thesis. By ignoring the statistic of the value of GamerID.

The important values and comparisons necessary are the “limit” values. This is due to the usage of these values in the game. Some of them (like MaxGameCountLimit), is important for setting up the requirement before a modification can happen. While others (like AverageCreepKillCount), set the grade of which some other values will further be modified. Additionally the Damage and Health values, are also important in setting boundaries for the available modifications.

It is quite clear that in general, odd has higher values than even. But to compensate for this, it is also visible in some areas that odd has a larger value difference between the Min and Max values than even. In conclusion, the odd control group contains more extreme values. Which made it the experimental control group of them both.
The History graph (see figure 6.2), is a visualization of the accumulated data, received from the test sequence. It includes handpicked GamerIDs seen on the y-axis. With the representing history results for each one of them. In short, all of the represented values are important in the modification part of the game. For instance, from the look of the equation below:

\[
\text{MaxPlayerHealth} = \frac{\text{HealCount}}{\text{GameCount}} \div (\text{AverageCreepKillCount} + \text{AverageBossKillCount}) \times \text{PlayerMaxHealthLimit} \times \text{GameCount}
\]

The value for the player’s max health is calculated with the History’s values \text{AverageCreepKillCount}, \text{AverageBossKillCount} and \text{GameCount}.

In addition, these values yield a better result in modification and balance for the player, as the values grow.

From the representation, the player with the GamerID 4 played significantly more than the rest of the group, giving this player higher possible values in the rest of the fields. This may be, due to the even control group, that the player was a member of.
The Players graph (see figure 6.3), represents the comparison of the results from the Player table. By showing handpicked GamerIDs, in the form of x-axis “PlayerXX”. The values MaxHealth, SwordPhysicalDamage and BlowgunPhysicalDamage. All depend on the Session, History and the control group it belongs to. For instance:

\[
\frac{\text{PlayerSwordDamage}}{\text{PlayerGunDamage}} \text{ (Depending on which one was used more)} = \frac{\text{oldSwordDamage} - \text{oldGunDamage}}{\text{oldGunDamage} - \text{oldSwordDamage} + \Delta \text{UsageCount}} \\
\times \left( \frac{\text{oldGunDamage} - \text{oldSwordDamage}}{1 + \Delta \text{UsageCount}} \right)
\]

In this case, the PlayerSwordDamage (from the equation) is the same as the SwordPhysicalDamage and PlayerGunDamage is the same as BlowgunPhysicalDamage from the table Player. In the equation the variables “oldSwordDamage” and the “DeltaUsageCount”, derive from the tables History and Session respectively.
From the resulting graphs (see figure 6.3, 6.4 and 6.5). By taking out the gamers stats in all three graphs and comparing each gamers’ results with regard to the players History and Settings. The players with the easiest game settings (i.e. high player max health with low enemy max health), had the most difficulty with the game’s mechanics because they had not played enough.

That being said. One can from this make the assumption that the players: Player1 and 3, Creeps and Bosses: ID1 and 3 (they are the same players), had the most difficult with the game’s mechanics. This assumption is due to the resulting values that these players had to their advantages compared to the other players’ values. For instance both player 1 and 3 had their PlayerMaxHealth values around 140 while the others was beneath 120. Both of them, had an over average sword and blowgun damage compared to the rest. Additionally while these players had an advantage in the form of, better player values. Still they in addition still had their obstacles, creep and boss still at the unchanged value.

On the other hand, for these algorithms to be true to their nature, they need to work on the other way around too. For the gamers that had it the easiest with figuring out the game’s mechanics and adapting to them. These gamers should have values showing more game difficulty (i.e. low player max health with high enemy max health).

An example where this assumption is held true, is by the examination of the players with the id 5 and 9. By the look from the History table, these players had a significantly higher score than the players 1 and 3, leading to the difficult settings for these players. With PlayerMaxHealth values around 100 while CreepMaxHealth on 60 and CreepPhysicalDamage around 16.
7 Discussion

This thesis has been demanding when it came to designing the game telemetry manager. The difficulty lay in the form of new programming languages and APIs used. The ASP.NET @Razor language for managing both front and back-end of the web application. With the added MySQL.Data.dll library for easy communication with the database and the Bootstrap plugin for a more dynamic feel to the web pages and its contents. The setup of database and working in MyPhpAdmin for managing the database with MySQL.

The client side made this thesis so much more complicated with constructing new designs for a suitable client based manager, both for the outgoing communication and for the reply response for managing the storing of metrics locally. An example of this was during the beginning period of the thesis xml was implemented to the game platform to manage the local storing. While not knowing that xml files could not be manipulated from multiple users, (from the database to store new data and from the game to retrieve the same data). This error led to the creation of multiple xml files with only version differences.

From this to a simple LocalData manager in unity to manage a number of variables was a far better solution. Getting rid of the problem with multiple instances of the files and complicated protocols for storing and retrieving sorted data from encrypted xml files.

Much input in the form of ideas were also received from interested people including the participants with new ideas of how game telemetry could be more efficiently utilized in, the context of the platform and in other cases. With ideas of both removing metrics such as the shield weapon and the stamina feature from the metric system, due to the lack of in game relevance. Additionally how game telemetry could be used in future releases and how that would affect the products and the communities surrounding it.

All this only showed that game telemetry may have been in the market of mobile applications, games, software and services. But it was reaching the epicenter of customer support and product personalization, with a new meaning due to the recent popularity of cloud services.

The question still lies in the matter of security and violations of personal information that came with it. The method only worked if the costumer agreed to the terms of use. For the service and it would be more normal for companies and product owners to start warning the costumers of what their services actually provided. Because of the fact that remote monitoring in its form could still be quite dangerous if done with ill intent.

In the end what could have been done differently for a better result lied in the choices made during the planning period. The choice of using xml was a total waste of time. The game server could have been much more efficient if it instead of being a web application, being a real game server. That could constantly run in the background. Else, remove the
whole thing all together, and just have the client side and the database as the web application never filled up any use in this project. It was only a way of showing of what possibilities lied ahead if used correctly.

On another subject A/B testing was a great choice for managing multiple outcomes from testers. This gave the results more depth, with available comparisons and paved the way for further improvements in settings.
8 Conclusion

When taking the results into consideration, it was difficult to make up one's mind if it in the end led to a success or a failure. This was due to the lack of relevant data. An example of this phenomenon was that the database management and the game telemetry in a whole, went well and in that case was a success. Although, due to the immense possibilities of metrics available for analyzation it was difficult to further improve the game to its best potential.

After a while of thinking and working with the telemetry, it became obvious that there were multiple ways to improve the game if the necessary mechanics had been available. It was clear that the principle and reason why game telemetry was so famous in the game industry was because of the fact that it requires a solid foundation to flourish. If the foundation was crumbly and not tended to the feature would simply mold, become scrap code that isn’t even used, instead only was in the way of important production.

Game telemetry is not meant to save a game that was not finished. It is for a game that needs that extra kick to get that chosen targeted audiences attention. It helps bring a new form of balancing to the game’s mechanics, through costumer/community feedback. If done well, it could be the icing on the cake for a simple game to become a competition for similar games on the market. An example of this is the mobile game Color Switch made by Marc Lejeune. It is a simple click game that in conjunction with game telemetry monetarizes metrics of where the player died and of what kind of obstacle, to further make the game more personal by adding the same obstacle to the player’s game. If the player loses on that obstacle frequently.

Though in short the thesis was a success, in the form of finding out the best possible control group for the targeted audience that balances the game mechanics the best, but also in the form of input from users and interested people. The goal of the thesis was to create a game telemetry system that could handle multiple users through A/B testing and change their personal experiences based on their personal ways of playing the game.

This was done by designing a relation based database that could handle multiple GamerID, SessionIDs and multiple controls groups. A web application that could manage the database through web page input and a query based MySQL protocols. Finally modifying an already existing, in production state game. With database management features; a login feature and in game player-choice analysis feature. With the ultimate goal of improving or finding ways to improve the UX through the method of game telemetry.
9 References

   http://blog.gameanalytics.com/blog/what-was-game-metrics.html
2. Gameanalytics: How To Analyze Games (2015-05-28)
   http://blog.gameanalytics.com/blog/how-to-analyze-games.html
   http://www.eldergame.com/2008/05/define-your-target-audience
4. Statista: Average Mobile App Session Length As Of 4th Quarter 2015
5. Gameanalytics: Games And Analytics: A Look Through Time (2015-02-26)
6. Gamasutra: Understanding Balance In Video Games By Keith Burgun
   https://www.optimizely.com/ab-testing/
8. Gameanalytics: Different Types of Game Metrics (2015-07-08)
   http://blog.gameanalytics.com/blog/metrics-all-game-developers-should-know.html
Appendix

10.1 Database Variables

10.1.1 Entities
1. GamerID - The index ID used as primary key for the rest of the tables
2. NickName – The name the tester signed in as

10.1.2 History
1. ID – History index ID
2. GamerID – Index ID from Entities
3. GameCount – Total amount of played sessions
4. GameTime – Total amount of counted in game time
5. HighestTraversedLevelCount – Highest reached level
6. OldSwordPhysicalDamage – The players previous sword damage
7. OldGunPhysicalDamage – The players previous gun damage
8. AverageCreepKillCount – An average of creep kills per session
9. AverageBossKillCount – An average of boss kills per session

10.1.3 Session
1. SessionID – Session index ID
2. GamerID – Index ID from Entities
3. LifeTime – The life time of the player for the session
4. SwordCount – Amount of times the player hit a target with the sword
5. GunCount – Amount of times the player hit a target with the gun
6. HealCount – Amount of times the player healed itself
7. TraversedLevelsCount – The Highest reached level
8. CreepKillCount – Amount of creeps killed
9. BossKillCount – Amount of bosses killed
10. CreepMaxHealth – The highest reached creep health during the session
11. BossMaxHealth – The highest reached boss health during the session
12. Date – The date that the data was added to the database

10.1.4 Player
1. PlayerID – Player index ID
2. GamerID – Index ID from Entities
3. MaxHealth – Current max health of the player
4. SwordPhysicalDamage – The swords physical damage
5. BlowGunPhysicalDamage – The gun physical damage
10.1.5 Creep
1. CreepID – Creep index ID
2. GamerID – Index ID from Entities
3. MaxHealth – Current max health of the creep
4. CreepClawPhysicalDamage – The creep claw physical damage

10.1.6 Boss
1. BossID – Boss index ID
2. GamerID – Index ID from Entities
3. MaxHealth – Current max health of the boss
4. BossFistPhysicalDamage – The boss fist physical damage
5. BossFireBallLauncherPhysicalDamage – The boss fire ball... physical damage
6. BossPillarOfFlamePhysicalDamage – The boss pillar of flame physical damage
7. BossJumpSpecialPhysicalDamage – The boss jump special physical damage

10.1.7 Settingseven/Settingsodd
1. GamerID – Index ID from Entities
2. Max/MinPlayerHealth – Max/Min player health limit
3. Max/MinCreepHealth - Max/Min creep health limit
4. Max/MinBossHealth – Max/Min boss health limit
5. Max/MinSwordDamage – Max/Min player sword damage limit
6. Max/MinGunDamage – Max/Min player gun damage limit
7. Max/MinCreepDamage – Max/Min creep claw damage limit
8. Max/MinBossFistDamage – Max/Min boss fist damage limit
9. Max/MinBossFireDamage – Max/Min boss fire ball launcher damage limit
10. Max/MinPillarOfFlameDamage – Max/Min boss pillar of flame damage limit
11. Max/MinJumpDamage – Max/Min boss jump special damage limit
12. AverageHealCountLimit – The limit of average heal count
13. MaxGameCountLimit – The limit of max games played
14. MaxTraversedLevelCountLimit – The limit record for level reached
15. AverageCreepKillCount – The average creeps killed on all sessions
16. AverageBossKillCount – The average bosses killed on all sessions
17. MaxCreepHealthIncrease – The amount of health the creeps receive for each kill
18. MaxBossHealthIncrease – The amount of health the bosses receive for each kill
10.2 LocalData Variables

1. maxHealth[] – Array holding the max health of all entities (player, creep, boss)
2. maxHealthLimit[] – Array holding the max health limit (settings)
3. minHealthLimit[] – Array holding the min health limit (settings)
4. localMaxHealth[] – Array holding the highest reached max health after a session
5. weaponPhysicalDamage[] – Array holding weapon damages (player, creep, boss)
6. maxHealthIncrease[] – Array holding the max health increase per kill (creep, boss)
7. lifeTime – Variable holding the session time
8. healCount – Variable holding amount of times the heal was used
9. traversedLevelsCount – Variable holding levels traversed
10. creepKillCount – Variable holding amount of creeps killed during session
11. bossKillCount – Variable holding amount of bosses killed during session
12. weaponUsageCount[] – Array holding counting times the player weapons was used