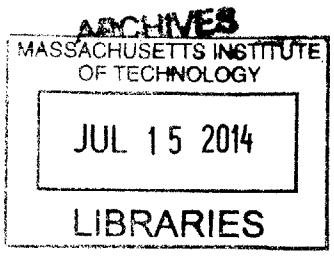
TaleBlazer Analytics: Automated Anonymous Analytics of Mobile Users' Behavior



by

Fidel Sosa

S.B, Massachusetts Institute of Technology, (2011)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2014

Signature redacted

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2014

Signature redacted

Certified by..... .....
Professor Eric Klopfer
Director, MIT Scheller Teacher Education Program
Thesis Supervisor

Signature redacted

Accepted by ....
Prof. Albert R. Meyer
Chairman, Masters of Engineering Thesis Committee

TaleBlazer Analytics: Automated Anonymous Analytics of Mobile Users' Behavior

by

Fidel Sosa

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2014, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

TaleBlazer is an augmented-reality platform that lets users create location-based games for their mobile devices. In order to determine the efficacy and use cases for TaleBlazer games, it is necessary to capture data about user behavior. This thesis presents TaleBlazer Analytics, an automated system which collects and analyzes mobile users' behavior in TaleBlazer games. It details the development of the TaleBlazer Analytics system, comprised of the backend data collection service and the front-end data analysis user interface.

Thesis Supervisor: Professor Eric Klopfer
Title: Director, MIT Scheller Teacher Education Program

Acknowledgments

I'd like to thank Eric Klopfer, Lisa Stump, and Judy Perry for giving me the opportunity to work on this project. It has been an eye-opening and rewarding experience that would not be possible without them.

I'd like to thank Lisa Stump for being a guiding force and great help during the project, as well as helping me determine a set of achievable goals for the project. I'd also like to thank Judy Perry for all the guidance and management she provided. This project would not have been as successful as it was without them.

I'd also like to thank the entire TaleBlazer Development team, including my fellow M.Engs Tanya Liu, Cristina Lozano, and Stephanie Chang. They provided much help in testing and making this project fun to work on.

I'd like to thank my academic supervisor, Boris Katz. Without his help during my undergraduate and graduate years, I would not be where I currently am. A great thanks to my friends, as well, who have supported me throughout the years.

Finally, I'd like to thank my family, whose sacrifices have allowed me to be where I am today.

Contents

List of Figures

Listings

Chapter 1

Introduction

The explosion of the mobile market has led to the proliferation of location-aware mobile devices and a wide range of mobile applications that provide location-based content. Augmented reality (AR) applications, for example, enhance the user's real-life environment with location-specific information. The prevalence and affordability of these mobile devices, such as smartphones and tablets, make them a natural choice as tools to augment education and learning. TaleBlazer is an augmented reality location-based game platform that allows users to create their own games that take place in the real world and play them on their mobile devices. A goal of the TaleBlazer project is to determine the educational impact of location-based games, such as how these games motivate users to learn more. TaleBlazer Analytics is an automated system that allows game designers and researchers to gather and analyze anonymous data about users' behaviors during TaleBlazer games.

1.1 Motivations for TaleBlazer Analytics

The wide gamut of user-created TaleBlazer games requires a data collection system that is both flexible and useful. TaleBlazer Analytics was developed to allow game designers and researchers to get specific metrics about when and how their games are played. For game designers, these metrics allow them to create more engaging and effective game experiences. For researchers, these metrics provide crucial insight into

users' gameplay progress and decision making. Game designers and researchers also need a way to quickly analyze data collected from game sessions.

Existing analytics solutions fail to provide data analytics relevant to TaleBlazer specifically and often do not provide an adequate level of user privacy. Furthermore, the unique nature of TaleBlazer games requires custom analysis of the generated data. As a result, it was necessary to develop TaleBlazer Analytics to meet our needs.

Over the past year and a half, I have developed an automated data collection system which seamlessly integrates with the existing TaleBlazer app and server architecture, working closely at each step with the TaleBlazer team. TaleBlazer Analytics is comprised of a backend server, mobile client, and web application; these individual components are jointly responsible for the collection, storage, and analysis of TaleBlazer gameplay data.

1.2 Chapter Summary

This thesis describes the background, design, and development of the TaleBlazer Analytics system. Chapter 2 details TaleBlazer in-depth and expands on the need for TaleBlazer Analytics. Chapter 3 explains the design process and preliminary work that was performed prior to the start of development. Chapter 4 details the TaleBlazer Analytics system and its component in depth. Chapter 5 goes into particular tests that were performed on the system and their effects on the project. Chapter 6 proposes future work for the project and Chapter 7 details the overall contributions of this thesis.

Chapter 2

Background

This chapter gives a background on the TaleBlazer project, including its separate components and how they work together as a whole. The history of TaleBlazer and past location-based projects are also detailed. Finally, the chapter expands on the need for TaleBlazer Analytics.

2.1 TaleBlazer

TaleBlazer is an augmented reality location-based game platform developed at the MIT Scheller Teacher Education Program (STEP). TaleBlazer is a platform in the true sense in that it is composed of multiple technologies which all come together to produce the TaleBlazer experience. At its core, TaleBlazer lets users create their own games that take place in the real world, using a web-based game editor. Users can then choose to publish their games to the world at large. Using the Android or iOS TaleBlazer mobile application, players play TaleBlazer games by downloading the game and physically walking around in the real-world location that the game takes place in. Players interact with virtual game agents: user-scripted entities that are placed by the game designer at specific GPS coordinates.

2.1.1 Overview of a TaleBlazer Game

A typical TaleBlazer game consists of the following parts:

- regions, which are the real-world locations where the game takes place
- roles, which encompass different sets of behaviors for the player(s) in the game
- scenarios, which encompass different versions of a game
- agents, which are in-game virtual entities that player(s) interact with
- traits, variables that belong to different in-game entities or the world
- scriptblocks, sets of programming instructions that define the behaviors for game entities

Regions

Regions are real-world locations where TaleBlazer games take place. Using the game editor, game designers define their game regions by selecting an area of a Google map. Regions define the area where in-game virtual entities, called agents, can be placed. Games can have one or more regions, each with their own name. The game designer also has the option of moving the player or agents from one region to another during the course of game using TaleBlazer's programming language. In order to play a TaleBlazer game, a player goes to the real-world location with their mobile device and moves around the region to activate or "bump" into the agents placed at nearby locations.

Roles

Roles allow game designers to define different characters or types of interactions for different sets of players. This allows designers to create role-playing game experiences. A game can have one or more roles. If a single role is defined, then all players experience the same game. Multiple roles let the game designer define different sets of behaviors for different roles. Each role has a name and an optional description,

which players see when choosing between roles at the start of a TaleBlazer game. For example, a game might have players choose between the roles of "Secret Agent" and "Police Officer".

Scenarios

Scenarios allow game designers to create different versions of the same game that players can choose between. For example, scenarios could be difficulty options, such as "Easy", "Medium", and "Hard". Games can have one or more scenarios, each with their own name. If a game has multiple scenarios, then the player is asked to choose between them at the start of a game. The game designer can tailor the behaviors of his game according to the choice of scenario.

Agents

Agents are the in-game virtual entities with which players interact during the course of a TaleBlazer game. Agents have a name, description, set of behaviors, and an optional real-world location. Agents could be anything from items like a treasure chest to a non-player character (NPC) that gives a player quests. Users have multiple methods by which they can interact with agents. Agents placed within a region have corresponding GPS coordinates. Players interact with the agent by arriving at the GPS coordinates of an agent. Some games allow the user to tap on an agent's icon on the map of the game to interact with the agent, referred to as "tap-to-visit". Although there are many different ways that a player could come to interact with an agent, an agent interaction in general is referred to as an agent bump.

Traits

Traits are variables that can be attached to agents, roles, or the game world in general. Traits have names and values and can be modified during the course of a game based on the behaviors defined by the game designer. Game designers can create their own custom traits, each with a user-defined value. Traits can be used to represent game mechanics, such as a score.

ScriptBlocks

ScriptBlocks is a block-based programming language that allows game designers to define the behaviors for their game. ScriptBlocks is block-based because all programming instructions come in the form of blocks that you connect together using a visual game editor (see Figure 2-1). For example, conditional behaviors can be defined using a conditional IF-ELSE block, which has sockets that allow you to insert other blocks to define the conditional statement to evaluate. Game designers can write game behaviors by creating agent or role-specific scripts, or by writing scripts applicable to the entire game world.

TaleBlazer contains a comprehensive set of programming blocks, encompassing a variety of purposes from logical and mathematical functions to TaleBlazer-specific commands. For example, a user can easily check the state of a game agent or move an agent or player from one location to another.

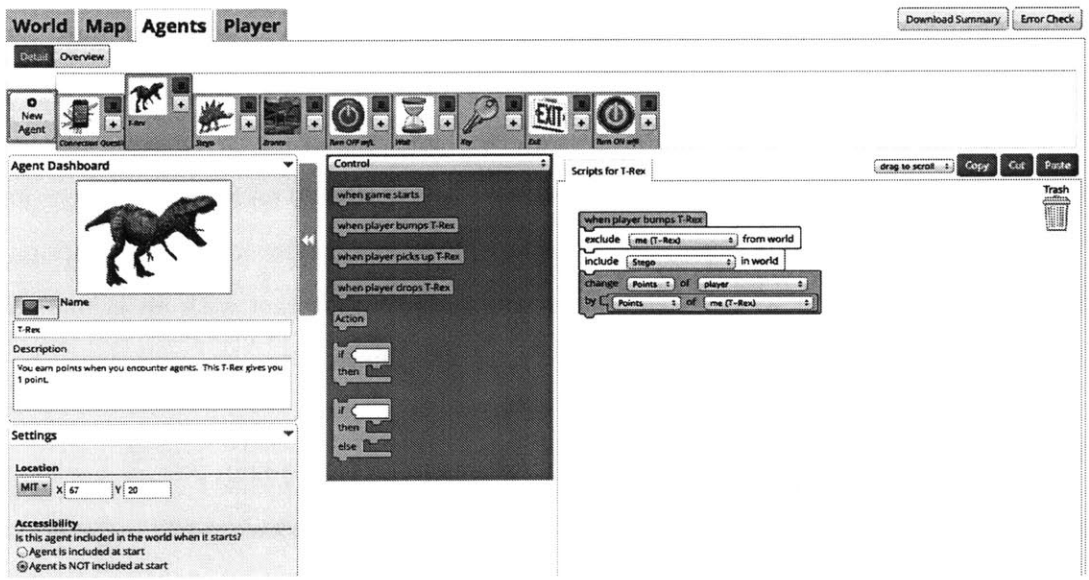


Figure 2-1: The TaleBlazer Game Editor web application

2.1.2 TaleBlazer Technology

The TaleBlazer platform is comprised of four main parts:

- the game editor, which lets users create their own games
- the mobile application, on which TaleBlazer games are played
- the repository server, which stores and serves games
- and the multiplayer server, which enables multiplayer TaleBlazer games

Game Editor

The game editor is an online web application that lets users create their own TaleBlazer games through the use of ScriptBlocks. Using the editor, users add and configure their game regions, create agents, roles and scenarios, and program the game by composing scripts. Users can save their games, which are then stored on TaleBlazer Server. The game editor also provides options for modifying the game interface that players see when playing a game on their mobile device. The game editor is written in JavaScript.

TaleBlazer Mobile

TaleBlazer Mobile is an Android and iOS mobile application that lets users play TaleBlazer games. The app lets users download TaleBlazer games onto their device and play them, utilizing the location-aware functionalities of the device. TaleBlazer Mobile interprets the blocks in each game file and executes them during the course of the game.

The TaleBlazer Mobile game interface consists primarily of a map of the current game region and icons indicating the position of the player and active game agents (see Figure 2-2). Tabs along the top of the screen provide different gameplay functionalities. For example, a game can include the Inventory tab, which tracks currently held items, or the Log tab, which contains a log of all the actions a player has taken during the course of a game.

TaleBlazer Mobile is written using Appcelerator Titanium, an SDK that lets developer write native Android and iOS applications using JavaScript.

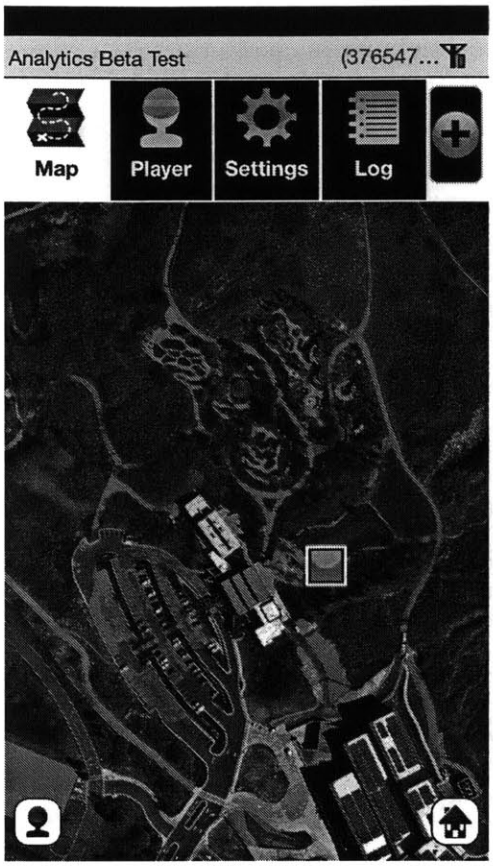


Figure 2-2: The TaleBlazer Mobile Main Map Interface

TaleBlazer Server

TaleBlazer Server is the main repository server, which handles user accounts, hosts the game editor, and stores and serves TaleBlazer games and game-related files (e.g. images, video). TaleBlazer Server is written in PHP using the CakePHP framework and backed by a MySQL database.

TaleBlazer Multiplayer

TaleBlazer Multiplayer is a separate server that provides multiplayer functionality for TaleBlazer games. It implements a separate protocol that synchronizes the state of

the world between all devices playing a multiplayer game. The server is written in Node.js, using JavaScript.

2.1.3 Past Projects

TaleBlazer is the most recent iteration and study into AR games performed at the STEP lab. Past projects, such as MITAR and StarLogo TNG, provided a basis on which TaleBlazer was built, namely the emphasis on augmented reality and the use of a block-based scripting language.

MITAR

MITAR (MIT Augmented Reality) was the immediate ancestor of TaleBlazer. Similar to TaleBlazer, MITAR sought to let users play location-based augmented reality games on earlier mobile platforms, such as Windows Mobile. MITAR also focused on the educational impact of augmented reality games on users. One game, called "Environmental Detectives", put users into the roles of investigators searching for the source of a toxic spill, taking measurements in order to determine the environmental impact. [5]

StarLogo TNG

StarLogo TNG (The Next Generation) was a project in programmable simulation modeling which allowed users to explore the workings of complex decentralized systems, such as ant colonies and traffic jams. [6] Similar to TaleBlazer, users could program the simulation using a block-based scripting language.

2.2 Why Build TaleBlazer Analytics?

The educational focus of the TaleBlazer project requires that the impact of TaleBlazer games on learning be measured. Game designers and educational researchers are supremely interested in seeing exactly how players play their games in order to draw conclusions as to their games' success and impact.

The rationale behind building TaleBlazer Analytics is two-fold. First, the public release of TaleBlazer requires an automated way of gathering quantitative data about all gameplay sessions. Previously, the only way to gather data was by observing players as they played. Second, existing analytics solutions fail to provide analytics relevant to TaleBlazer with a desirable level of privacy. To this end, it was necessary to build TaleBlazer Analytics to meet our requirements.

2.2.1 Purpose of Data Collection

The overall purpose of collecting TaleBlazer gameplay metrics is to provide interested parties with information to make informed decisions regarding the effectiveness of their game, across different aspects. Specifically, these parties are interested in quantitative metrics, such as the number of players that completed a game and the choices that were made during a gameplay session.

This type of data could be used to identify buggy game scripts or points of player confusion. It could also be used to determine the appeal of specific narrative elements. For example, a game designer would be able to determine if changing the name of a role from "Police Officer" to "Officer Awesome" made the role more appealing.

There are four main parties that are interested in this type of data:

- Game designers
- Educational researchers
- TaleBlazer developers
- Institutions

Game designers

Game designers are primarily interested in seeing how players progress through the game and the choices they make along the way. Using this information, a game designer can quickly identify problematic spots. For example, players may stop playing after a particular point in the game because the instructions to proceed aren't clear

or there is a bug in the game scripts. As a result, the game designer can improve the game to make it a better experience for the players.

Educational researchers

Educational researchers are interested in seeing how a game's content affects a player in the short or long-term. The choices that a player makes during a session can inform the researcher as to the level of a player's knowledge or how the content affected the player's understanding of the topic at hand. The gameplay metrics can be paired with external data, such as post-gameplay questionnaires or interviews. For example, a researcher studying a game about the environment might look at if a player encountered an EPA agent in-game or how fast they completed the game to see if a player missed crucial information or may not have been paying attention. Games can also include in-game research questions, asking players' about their motivation for future activities and their interest in certain educational topics.

TaleBlazer developers

TaleBlazer developers are interested in the technical aspects of games in order to inform future technical decisions and feature roadmaps. For example, the kinds of devices being used and the version of their operating systems (OS) are supremely useful in determining possible technical issues related to specific devices and the adoption rate of new OS versions. This information can then be applied to guide the TaleBlazer development process and provide concrete data with which to prioritize tasks.

Institutions

Analytics data may also be used for purposes outside of games. TaleBlazer is currently in use at several institutions across the country, such as botanical gardens, zoos, and historical sites. In these cases, quantitative metadata concerning when and how long games are played can prove especially useful in determining the effectiveness

and impact of TaleBlazer games on visitors. Additionally, it can also help provide information about the impact of exhibits and areas of an institution.

Institutions may be particularly interested in raw numbers, such as how many visitors played a particular game and the game's popularity over time. This can help institutions determine how funding gets spent to improve exhibits and areas. Furthermore, it can also help them identify ways to reach particular hard-to-reach audiences, such as "tweens".

2.2.2 Motivations

The user-generated nature of TaleBlazer games results in games that span a huge range of possibilities. As a result, TaleBlazer requires an analytics solution that is both *comprehensive* and *useful*: comprehensive in order to accommodate the range of possibilities of TaleBlazer games, and useful in order to provide meaningful and relevant analytics. In particular, an automated and non-interfering data collection system was necessary in order to collect comprehensive data; a custom analytics system was necessary in order to provide statistics custom and specific to TaleBlazer games.

Automated Non-Interfering Data Collection

With the public release of TaleBlazer, it was necessary to automate the collection of data from all gameplay sessions. Previously, all TaleBlazer games were played in a moderated setting, in which adult facilitators would guide gameplay, troubleshoot, and see how the game was being played in real-time. This approach is infeasible when groups are unmoderated or large. As a result, it was necessary to develop an automated system for collecting gameplay data for all sessions.

The previous method for collecting gameplay data involved observing players as they played. Although this approach yielded (and continues to yield) important data regarding players' dialogues and moods, it had two downsides. First, it resulted in an incomplete picture of an entire group's gameplay sessions, as the number of players

that could be observed was limited to the number of facilitators present. Second, it interfered with gameplay. Players tended to act completely differently when being observed than when left to their own devices, resulting in skewed data. As such, an automated system was necessary in order to gather quantitative data about all gameplay sessions without interrupting players and to complement existing manual observation methods.

Relevant Analytics and Privacy

Existing analytics services were investigated to see if they fit the needs of the TaleBlazer project. These services focus on providing a general data collection solution for its users. Solutions provided by Flurry and Mixpanel, for example, focus on an event-based method of analytics, which tracks unique events across every use of the app. However, these services cannot generate data analytics that are useful and specific to TaleBlazer because they focus on collecting data rather than providing application-specific data analysis. As a result, it was necessary to develop an analytics system built with TaleBlazer in mind. Specifically, this means that features such as data collection, categorization, and statistics calculation can be customized to fit the use cases of TaleBlazer Analytics users.

A separate concern arose when dealing with the nature of the TaleBlazer analytics data and the question of privacy. Privacy is supremely important to the TaleBlazer project, as games are often played by minors and students. As a result, it is a requirement that any collected data be completely anonymized and only used for educational and research purposes. Existing solutions, such as Flurry, do not guarantee the absolute privacy of the data provided to them and in fact may share that data with third parties. [1] As such, it was necessary to build TaleBlazer Analytics to ensure that data was anonymized and used only for the purposes of the TaleBlazer project.

Chapter 3

Preliminary Work

Prior to the start of the development of TaleBlazer Analytics, many decisions had to be made in order to arrive at a feature specification for the system. In order to develop a useful analytics system, the types of analytics data to capture had to be determined. To meet the technical needs of the project, the technology used to build the new system had to be benchmarked and chosen. Finally, the user interface for the analytics site was mocked up and underwent an iterative design process.

3.1 Types of Analytics Data to Capture

The key purpose of TaleBlazer Analytics is to capture metrics about TaleBlazer games. In order to achieve this, it was first necessary to determine exactly what kinds of data would be useful to capture and if we could capture it. First, preliminary work was performed in TaleBlazer Mobile to identify the types of information that the system was to collect through the development of an additional mobile tab dedicated to logging information. Second, a collaborative process was undertaken to arrive at user stories for TaleBlazer Analytics users. Finally, these user stories defined the analytics data that was to be captured.

3.1.1 Log Tab

The Log Tab is a new tab in TaleBlazer Mobile that was developed specifically to lay the foundation for the development of the TA system. This tab contains a chronologically sorted list of game events that players can view in order to get a high-level picture of the actions they have taken during a game. The Log Tab contains information such as when a particular agent was bumped and whether a player picked up certain items.

The purpose of the Log Tab was three-fold. First, it allowed players to review their actions during a game, which was particularly useful for long games. Second, it served as a way to identify game events that would be useful to collect as data. If a particular event was deemed necessary to go in the log, then that type of data was prioritized to collect. It also helped alert us to the kinds of data that were possible to collect. This helped us reach a feasible and grounded feature specification later on. Second, it served as a way to introduce players and game designers to the types of game events that we would later go on to track in TaleBlazer Analytics.

3.1.2 User Stories

In order to determine the specific events that were to be captured, we worked backward to determine what types of analytics data would be the most useful to users. This came in the form of user stories: short sentences that describe at a high-level what users will want from a product. The types of users that would be interested in TaleBlazer Analytics were decided and short stories were written for each of them. The three types of users that were determined were:

- occasional users
- power users (i.e. designers and researchers)
- TaleBlazer staff (i.e. developers and researchers)

Occasional users are people that have made and played a few TaleBlazer games and are primarily interested in high level analytics data, such as how long people played

their game and how many people completed the game. Power users are game designers and educational researchers and are interested in more detailed statistics, such as the ability to categorize the data based on the roles and scenarios of players. These users are also interested in generating their own custom analytics data. Finally, TaleBlazer staff are developers and researchers that are interested in technical information such as the models of devices being used and their screen resolutions. One of the user stories we wrote, for example, was the following: "As an occasional user, I want to be able to see what version of my game they played."

3.1.3 Analytics Data

With the user stories, we were able to determine exactly what kinds of data we wanted to collect. TaleBlazer Analytics takes an event-based approach to data collection. All events record the time that they occurred in-game, as well as event-specific information. The data that we were interested in capturing involved the following:

- devices
- sessions
- agent bump events
- region switch events
- game completion events
- custom events

Devices

Device information was critical to capture in order to gather technical metrics and to be able to identify unique users. This information involved the OS and its version, the model of the device, and the screen resolution. Additionally, each device has a unique ID specific to TaleBlazer and cannot be traced back to the device or used to identify it for non-TaleBlazer purposes.

For power users, device information can be used to analyze players' behaviors throughout multiple games. For TaleBlazer developers, this information can help prioritize development tasks and identify device-specific issues.

Sessions

Sessions represent all the information about a single gameplay session. All events that take place within a game are tied to a particular session. Sessions consist of the time that the game was started and the time of the last event that occurred. They also contain information about the particular role and scenario chosen for that gameplay session and whether the tap-to-visit setting was enabled. Finally, the session is tied back to the particular version of the game being played and the device on which it was played.

In general, sessions provide the basis for all analytics data, as they can be used to determine information from the number of overall players to the sequence of actions that a player took during a gameplay session.

Agent Bump Event

An agent bump event occurs when the player "bumps" into an agent, via a variety of methods. An agent can be bumped by:

- walking within range of its GPS coordinates
- being tapped on when tap-to-visit is enabled
- being encountered via the augmented reality camera HUD
- unlocked by entering a password called a clue code
- being accessed from the inventory

Each agent bump event records the name and unique ID of the particular agent that was bumped. It also details how the agent was bumped and the session that the event took place in.

Agent bumps can be used to identify information such as the number of unique times that an agent was bumped and how far a player made it into the game.

Region Switch Event

A region switch event occurs when the player moves from one game region to another, triggered by a "Move To Region" block. Each region switch event records the name and unique ID of the region, as well as its corresponding session. With this data, region switch events can be used to determine information such as the number of players that reached a certain point in a game.

Game Completion Event

Game completion events record when a particular game was completed. Prior to TaleBlazer Analytics, TaleBlazer games did not have a fixed concept of the end of a game. In order to track this data, a new block was added to the game editor called the "End Game" block. The sole purpose of this block is to mark the end of a game from an analytics standpoint. Game designers can use this block to mark a point after which they do not want to continue tracking events for that session. For example, a game might allow players to replay the game, which the game designer might not wish to track events for. Each game completion is tied to a session.

Custom Events

Custom events allow game designers to track player choices or dynamic values custom to their specific games. In order to accomplish this, a new block was developed called the "Analytics Event" block. Users can create and name their own custom event and track whichever values they would like. For example, a user might create an analytics event called "Player score" and track the value of a trait representing the player's score. The game designer then places the block where they would like the event to be triggered. This gives game designers a flexible way of tracking data specific to their games. For example, custom events could also be used to track the actions of a player, such as whether a player picked up an anvil.

In terms of TaleBlazer Analytics, custom events record the name and unique ID of the custom event, as well as the value the game designer wanted to track. Custom events are tied to their corresponding session.

3.2 Choice of Server Technology

In order to build a robust system, it was necessary to determine the technical requirements of the project, driven by our goal to collect extensive analytics data. These technical requirements led to the consideration of two different server technologies: Node.js and PHP/Apache.

3.2.1 Technical Requirements

The main goal of TaleBlazer Analytics was to provide real-time data collection and analysis for its users. Furthermore, a goal of the project was to provide a fine level of detail and granularity in the analytics data. For example, users of the system should be able to view the number of unique and total bumps for a particular agent, categorized by data such as the role of the player and the date. This fine level of detail required TaleBlazer Analytics to keep track of the unique events that occur for the types of data that we wanted to capture. In order to accomplish this, each individual event would have to be processed and stored. As a result, TaleBlazer Analytics required a server technology that could handle a massive amount of concurrent requests.

Another requirement had to do with the existing technologies in use on the TaleBlazer project. In general, the choice of technology for a new project is driven largely by the existing technologies already in use. This allows members of the team to more easily transition and understand the multiple components in use on the project. The majority of the TaleBlazer platform is written in JavaScript, with the sole exception being TaleBlazer Server, written in PHP. As a result, it was necessary to pick a server technology that utilized either JavaScript or PHP in order to maintain technical consistency throughout the TaleBlazer platform.

3.2.2 Node.js vs. PHP

The two server technologies that met our requirements were Node.js and PHP running on Apache (referred to as PHP/Apache). These technologies were already being used in the TaleBlazer platform: TaleBlazer Multiplayer runs on Node.js and TaleBlazer Server runs on PHP/Apache.

Node.js

Node.js is an asynchronous, event-driven software platform for building highly scalable network applications in JavaScript. Node utilizes the Google V8 JavaScript engine for its runtime, which is also used in the Google Chrome browser.

Typical multi-threaded servers allocate a thread per request, which results in high memory overhead. For example, at a typical 2MB per thread, a server with 6GB of RAM would theoretically be able to handle 3000 requests, ignoring all other processes and operations. Node's main benefit is that it runs all operations asynchronously on a single thread, using non-blocking I/O operations. As Node runs on a single thread, the memory overhead for a server running Node is significantly less than that of a typical multi-threaded server. Node's asynchronous event-loop means that massive amounts of concurrent requests can be handled. This is because all requests are handled asynchronously with non-blocking I/O, so no request blocks another from completing. [8]

PHP/Apache

The second choice for server technology was PHP running on Apache. The existing TaleBlazer Server utilizes this stack, using CakePHP as its Model-View-Controller (MVC) web application framework. Apache is one of the most widely used open source servers on the Internet.

Benchmark Methodology

One of the main requirements for TaleBlazer Analytics was to be able to handle large numbers of concurrent requests, generated by the TaleBlazer Mobile clients sending back analytics data. In order to determine the best choice of technology, it was necessary to create a benchmark for comparing the speeds of Node.js and PHP/Apache. Two servers were written in Node.js and Apache, each exposing a single API endpoint. Each server would insert a row filled with random information into a MySQL database table on HTTP requests to the API. Each server was deployed onto an Amazon EC2 m1.small instance with 1.7 GB of RAM.

The benchmark utilized Apache Bench to determine the number of requests per second each server could handle at different numbers of concurrent requests. Each server was tested 10 times at each level of concurrent requests. The levels of concurrent requests started at 100 and increased by 100 until 900 concurrent requests were reached. The results were then averaged over 10 runs of the benchmark.

Benchmark Results

The benchmark showed that Node was able to achieve an average of 505 requests per second, with a max of 762 req/s @ 100 concurrent requests and a min of 140 req/s @ 900 concurrent requests. PHP/Apache was able to achieve an average of 237 requests per second with a max of 220 req/s @ 100 concurrent requests and 80 req/s @ 900 concurrent requests.

The purpose of this benchmark was to get a sense for the difference in speeds between the two server technologies. The results of this benchmark pushed the project to decide on Node.js as the server technology.

3.3 UI Design

In order to get a better sense of how to present useful data analytics to TaleBlazer Analytics users, it was necessary to mock up the user interface for the site component of TaleBlazer Analytics. Existing analytics dashboards were investigated and several

mockups of increasing fidelity were then created. The mockups were presented to TaleBlazer partner institutions who provided feedback on the designs. Figure 3-1 shows one of these early mockups.

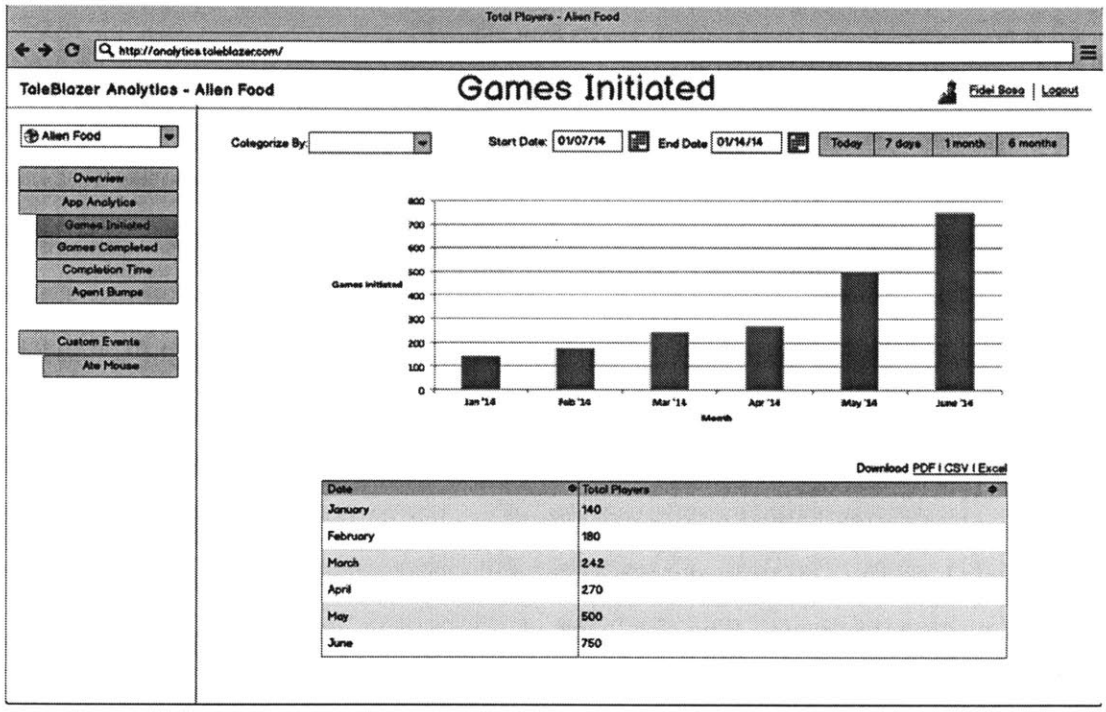


Figure 3-1: Early design for the Games Initiated page, which showed the number of games that were started. This would eventually become the Games Played page in the final site.

3.3.1 Effect of Mockups on Project Requirements

In designing the mockups, we were able to determine the set of pages that comprise the TaleBlazer Analytics site. As a direct result, this informed the technical specifications of the project as to the specific types of statistics that would need to be generated from the collected data. For example, early mockups introduced the idea of categorizing information. This feature would go on to become the categorization function, which is an integral part of all statistic calculations in the final TaleBlazer Analytics system.

From investigating other existing analytics dashboards, features such as fast date

range filters were introduced, which live in the final system. The mockups that were created also included features for the future of TaleBlazer Analytics, such as detailed data visualizations.

3.3.2 Partner Feedback

One of the main goals of creating these mockups was to familiarize our partner institutions with the analytics interface and the functions that would be available to them. In turn, this gave our partners the opportunity to give us feedback regarding desired features, improvements, and changes.

To this end, annotated mockups were sent to our partner institutions along with a short survey. The survey asked our partners to rank the site pages that they foresaw themselves using the most, as well as the categorization options they would most use. This information helped us prioritize development on the most requested pages. Features such as the ability to download the full set of analytics data were a direct result of partner feedback. This feedback also helped us determine how analytics users would use the site. One respondent answered that during game development, they would focus on how often agents were bumped and if the game was completed to help them pinpoint trouble spots. Once the game was stable, they would focus more on how often and when people were playing the game.

Chapter 4

TaleBlazer Analytics

This chapter provides a high-level overview of the three different components that make up the TaleBlazer Analytics system: the analytics server, client, and website. The TaleBlazer Analytics system is described in its entirety, including specifics about each component and how the components work together to form the core of the analytics functionality.

4.1 System Overview

TaleBlazer Analytics is composed of three different components that work together to gather, analyze, and present gameplay metrics for TaleBlazer games. The three components are the:

- analytics server
- analytics client
- analytics website

The analytics server is a Node.js application that is responsible for receiving, processing, and analyzing gameplay metrics, as well as serving the analytics website. It forms the backbone of the system.

The analytics client is a standalone JavaScript client which is integrated into TaleBlazer Mobile. It is responsible for the actual collection of gameplay metrics and handles all the interactions between TaleBlazer Mobile and the analytics server.

The analytics website allows users to view and download the calculated statistics for their TaleBlazer games. The site receives its information via calls to an API (Application Programming Interface) hosted by the analytics server. The site is written in JavaScript with a focus on client-side page rendering, with light server-side templating.

4.2 Analytics Server

This section provides an in-depth explanation of the technology and development process behind the analytics server.

4.2.1 Technical Overview

The TaleBlazer Analytics server is a Node.js web application that is responsible for collecting, processing, and analyzing all the gameplay metrics received from TaleBlazer Mobile via the analytics client. The analytics server is built using Express, a web application framework that provides a robust library for building web services. The server follows the MVC development pattern for structuring the application. The server is a RESTful web service, which means that all external interactions with the server occur via REST APIs. MySQL is used as the backing database for analytics data, in order to integrate with the existing TaleBlazer Server database.

Strict development methodologies were adopted to ensure that the server is easily modifiable, extensible, and maintainable. To this end, the analytics server is testable and extensively documented. The server was built to be easily configurable and simple to deploy in local, testing, and production environments. Deployment in production environments alongside TaleBlazer Server is accomplished through the use of Nginx as a proxy server.

4.2.2 Server Structure

A crucial step in developing the analytics server was deciding on the best way to structure the server. The main goal was to reach an optimum level of component decoupling, which would allow features to be easily implemented and modified without affecting unrelated parts of the application. To this end, the Express web application framework was used to provide the high-level web-oriented functionality, such as routing and HTTP request handling. The application is split up into models, views, and controller, which separate code according to their functionality.

Express

Express is a light Node framework for building web applications. It is built on top of the low-level Node.js HTTP module and provides a high-level API for handling all interactions having to do with HTTP requests. At its core, Express provides simple ways of routing URLs, parsing HTTP requests, sending HTTP responses, and defining paths for request processing via middleware. Defining what code to execute based on a request to a URL (i.e. routing) is simple using Express.

```
1 // Responds to a GET request at /helloWorld with the text 'Hey!'
2 app.get('/helloWorld', function(res, req) {
3   res.send('Hey!');
4 });
```

Listing 4.1: Example of Express' URL routing

Express handles incoming requests by passing a request along a defined path of functions, each known as middleware. For the analytics server, this provided the ability to implement robust error handling and logging functionality.

Model-View Controller

Code in the analytics server is organized according to the Model-View-Controller (MVC) pattern, which allows us to separate code into logical components based on their functionality. Models represent database objects and contain retrieval and mod-

ification methods. Controllers handle incoming requests by retrieving information from specific models and sending back a response. Views contain the logic for the user interface and request information from controllers.

Models Models in the analytics server correspond directly to their respective database tables and allow us to easily perform queries on the database without having to write SQL. This is accomplished via Sequelize, an object-relational mapping (ORM) library, which connects to the database and abstracts SQL queries and relations between tables by providing a high-level JavaScript API. Sequelize also provides migration functionality, which lets us make incremental changes to the database schema. Additionally, it performs model validation prior to making any database calls for an extra level of security. The models for TaleBlazer Server correspond directly to the types of analytics data in Section 3.1.3.

Controllers Controllers for the analytics server handle, process, and respond to requests. Each controller provides the functions that get executed when a URL is requested. For example, requests to register or get information about a device will always go through the device controller.

Views The analytics server also serves the analytics site and as such is responsible for providing the HTML, JavaScript, and CSS resources required for each page of the website. Dynamic page content for the analytics site is largely performed by client-side JavaScript. However, a minimal amount of HTML template rendering is performed on the server. Views correspond directly to these HTML templates, which contain the markup for the page layout. The ECT JavaScript template engine powers the template rendering and allows us to split up templates into logical components, thereby making it easier to modify the pages of the analytics site.

4.2.3 REST API

All external interactions with the analytics server occur through a REST API. All API endpoints return JSON (JavaScript Object Notation) and conform to a standard response format. For consistency and standards-compliance, each endpoint enforces that the correct HTTP headers be set and responds with the correct HTTP status code for the state of the response. The analytics server API endpoints are divided into two groups:

- the data collection API, used to collect data from TaleBlazer Mobile
- the data analysis API, used to provide statistics for the analytics site

What is REST?

REST, standing for Representational State Transfer, is a style for building HTTP APIs, using the HTTP verbs (GET, POST, PUT, DELETE) as actions on resources. [7] A resource can be thought of as a noun (e.g. the device resource). Basic creation, modification and read operations can be performed by making a request to a resource using a particular format. For example, a GET request to `http://SITE/device/4` would return a device with id equal to 4.

API Format

Each API endpoint responds with JSON formatted according to the JSEND format. [2] Each response object has a "status" key, which has a value of "success" or "error" depending on the outcome of the request. Successful responses include a "data" key, which contains data pertinent to the original request. Failure responses include a "message" key, containing a human-readable string describing the error. Conforming to a standard response format makes it simpler to parse responses to API requests and maintains consistency throughout the project. Listing 4.2 gives an example of the response body of a successful request to the device API.

```
{
    status : "success",
    data : {
        "device" : { "id" : 1, "model" : "iPhone 5", ...}
    }
}
```

Listing 4.2: Example API format, using JSEND

All API endpoints require correct HTTP headers for requests. For example, the `Content-Type` header, which defines the format of parameters in the request body, must be set to `application/json`, if the request body is in JSON format. Similarly, the `Accepts` header, which defines acceptable formats for the response, must be set to `application/json` in order for the response to return JSON. Again, this ensures that API requests behave consistently and correctly as inconsistent or undefined behavior often leads to bugs or exploits.

In addition to the status field in the JSON response, each endpoint also responds with the correct HTTP status codes. For example, a status code of 500 represents an internal server error. A status code of 201 represents that a resource was successfully created. These status codes provide an additional way to check the status of an API request.

Data Collection API

The data collection API consists of all API endpoints that are involved in the process of gathering gameplay metrics from TaleBlazer Mobile. The data collection API consists of three resources: devices, sessions, and events. With respect to the process of data collection, the following endpoints are the most important:

- Device Registration API
- Session Request API
- Batch Event Processing API

Device Registration The device registration API provides a way to register mobile devices with the analytics server. The endpoint takes a unique TaleBlazer Analytics ID and information about the device (as defined in Section 3.1.3). If successful, the API responds with a record of the newly registered device.

Session Request The session request API allows gameplay sessions to be recorded on the analytics server. It takes the device's unique TaleBlazer Analytics ID and information about the session (as defined in Section 3.1.3). If the device has not already been registered with the analytics server, then the API sends back a failure response. Otherwise, it responds with a record of the newly created session, including the unique session ID used to tie events and sessions together.

Batch Event Processing The batch event processing API is the most complex in the data collection group. The batch event API takes a list of event objects, each corresponding to the types of events previously mentioned (e.g. agent bumps or region switches) and saves each event to the database. If a single event in the list of events is invalid or an error occurs while saving an event, then the entire request fails and the API responds with a failure. This is accomplished through the use of database transactions, which provide an "all-or-nothing" guarantee: either everything is saved or nothing is. This is beneficial because it alerts the API user that an error has occurred and that no data was saved. Otherwise, it would be difficult to tell what was saved to the database and what wasn't. As a result, it provides consistent "fail-fast" behavior for the API and avoids tracking possibly problematic data.

Data Analysis API

The data analysis API consists of the API endpoints that calculate and provide analytics data for the analytics site. Currently, all statistics are calculated on-the-fly when requested via the API.

The data analytics API consists of the following endpoints:

- Overview API
- Games Played API
- Gameplay Duration API
- Agent Bumps API
- Custom Events API
- Raw Data API

Each API takes a start and end date representing the date range of data to analyze. Excluding the Overview API, they also take a categorization method, which categorizes the data based on one of the following:

- Date (Excluding the Agent Bumps API)
- Role
- Scenario
- Game Version
- Agent (Only for the Agent Bumps API)

Overview The Overview API provides key statistics concerning the overall performance of a game. The statistics that are calculated include the number of games initiated, the number of games completed, the average time users took to complete a game, and the lifetime number of downloads. This information is calculated purely from information on the Session model (see Section 3.1.3).

Games Played The Games Played API responds with information about the total number of games played. This is further broken down into non-overlapping groups:

- the number of games initiated (but not completed)
- the number of games completed

These statistics are calculated solely from information on the Session model, as in the Overview API.

Gameplay Duration The Gameplay Duration API provides statistics on the amount of time that people took to play a game. In particular, gameplay sessions are separated into buckets based on their gameplay time. Currently, sessions are bucketed into time ranges of 15 minutes, starting at 0 minutes and going up to 120 minutes. Listing 4.3 demonstrates a sample response.

```
{
  "status": "success",
  "data": {
    "results": [
      {
        "0-15": 3,
        "15-30": 4,
        "30-45": 0,
        "45-60": 6,
        "60-75": 3,
        "75-90": 5,
        "90-105": 1,
        "105-120": 4,
        "120+": 0,
        "role": 1,
        "entityName": "Explorer"
      }
]}}
```

Listing 4.3: Gameplay Duration API response, showing the number of sessions that fell within certain ranges of gameplay time, categorized by the Explorer role

As before, all information is directly calculated from information on the Session model.

Agent Bumps The Agent Bumps API provides information about the number of times all agents were bumped, divided into unique bumps and total bumps per agent. Unique bumps are calculated by determining if a particular agent was bumped at all during a session. Further bumps then contribute to the number of total bumps.

Custom Events The Custom Events API provides information about the number of unique and overall times that a custom event was triggered. A unique custom event is defined as whether a player triggered the custom event during gameplay. Further triggers of the custom event then count towards the overall count. Due to the fact that custom events can have any set of values, custom events are further grouped by their particular value, in addition to the normal categorization options.

Raw Data The Raw Data API is the sole exception to the standard JSON format. Instead of JSON, the Raw Data API responds with a CSV of all the raw event data for a particular game over a given date range. The purpose of this endpoint is to allow power users to work directly with the collected data to perform further analysis.

4.2.4 Development Methodology

In addition to the goal of collecting and analyzing gameplay data, a goal of the project was to develop a maintainable server that was easily modifiable and extensible in order to allow future developers to easily add features and continue building atop the platform. To this end, a test-driven methodology of development was adopted and stringent documentation standards were employed throughout development.

Test-Driven Development

The analytics server was developed according to the Test-Driven Development methodology. This methodology emphasizes that tests for logical chunks of functionality be

written first before starting development on the feature. As a result, the developer is forced to consider in-depth the behavior of the feature and the success and failure conditions. On completion of a feature, the tests are run and the feature deemed complete if all the tests pass. This results in a codebase with extensive test coverage. Furthermore, the overall test suite provides a way of verifying that changes to the codebase do not accidentally break functionality.

For the analytics server, this methodology resulted in extensive test coverage for the API. Tests were written using the Mocha testing framework and the SuperTest HTTP assertion library. These libraries emphasize readable and self-documenting tests. For example, Listing 4.4 shows how a test is written.

```
1  it('responds with json', function(done) {
2    request
3      .get('/session')
4      .set('Accept', 'application/json')
5      .expect('Content-Type', /json/)
6      .expect(200, done);
7  });
```

Listing 4.4: Example Mocha test, testing that the Session API responds in JSON

Documentation

Documentation for the analytics server was a continuous and simultaneous process alongside development. The overall goal was to write clear and useful documentation. The codebase was documented in two ways. First, the code was written to be as self-documenting as possible, using clear and descriptive variable names. Code linters and style formatters were employed consistently throughout the development process to enhance readability and ensure style consistency throughout the various files. Second, comments were added to pieces of code that required further explanation, detailing specific processes or ideas further. For example, the ideas behind statistic calculations were expanded on in comments.

In addition to the documentation inside the codebase, detailed step-by-step in-

structions were written explaining how to install and deploy the server. Documentation also included pre-built files for performing test API requests using an external utility and a file giving an overview of the database schema.

This documentation standard was also employed for the development of the analytics client and site.

4.2.5 Installation and Deployment

Configuration and Installation

To simplify future development and deployment, the server was built to be easily configurable and simple to deploy to any environment. Server configuration comes in the form of external JSON files and JavaScript modules that contain server and database configuration details. Example configuration files are provided to ease the setup process.

Installation and deployment to any environment is similarly simple. Developers simply checkout the project from a Git repository, fill in their configuration details, and run a single command to start the server. As a result, getting started developing for the analytics server is fast and easy.

Logging

Logging functionality was implemented into the server in order to simplify troubleshooting in production environments. Logging is handled via Winston, an asynchronous logging library for Node.js. To ease troubleshooting, error and request logs are saved to a log directory, configurable by the developer. Error logs contain stack-traces, server performance statistics, and additional error-related information.

Deployment alongside TaleBlazer Server

The TaleBlazer Analytics server was deployed on the same machines as the existing TaleBlazer server. Early testing with our partners indicated that certain networks would restrict access via HTTP to sites on ports other than port 80. As a result,

it was necessary to find a way to deploy TaleBlazer Server and the analytics server alongside each other and have both servers respond to requests at port 80. Nginx was deployed as a reverse proxy server to accomplish this.

Nginx is a high-performance HTTP server which is often used as a reverse proxy or load-balancer in front of other servers. A reverse proxy server is a server that retrieves information from other servers on behalf of a client making a request. In our particular case, Nginx allows TaleBlazer Server and the analytics server to both respond to requests on port 80. It accomplishes this by performing requests to the respective server on behalf of the original client and then serving the request. Fitting with the goals of the project, Nginx is also easily deployed and configured. To ease future deployment, the configuration and deployment process was extensively documented.

4.3 TaleBlazer Analytics Client

This section provides the technical details of how the TaleBlazer Analytics is built and how it interacts with the analytics server to collect gameplay data.

4.3.1 Technical Overview

The TaleBlazer Analytics client is a standalone JavaScript module that handles the API workflow for collecting and sending data from TaleBlazer Mobile. TaleBlazer Mobile implements the client's public API to track gameplay events. The client is then responsible for storing gameplay data and sending it to the analytics server.

The TaleBlazer Analytics client is designed to function in both offline and online capacities. TaleBlazer games can be played in offline situations, where internet connectivity is not available. In order to track this data, the analytics client stores all gameplay data locally until a data connection becomes available, at which point it sends the data to the server for storage.

4.3.2 API Workflow

The main job of the analytics client is to interact with the Data Collection API of the analytics server to track gameplay data gathered from TaleBlazer Mobile. To track data, a specific workflow has to be followed in order to ensure that data is collected from offline and online gameplay sessions. The general workflow is as follows:

First The mobile device is registered with the analytics server.

Second At the start of a new game, a session is requested from the analytics server.

Third Events are periodically sent to the analytics server in batches.

Each step in this workflow requires that the prior steps be successfully completed. If a step fails, the client automatically retries the failed step periodically. For example, if a device is not registered with the server and a new game is started, the client will automatically attempt to register the device before requesting a session from the analytics server. In the case of TaleBlazer games played offline or in low connectivity situations, the client generates local sessions to be able to track events. The above workflow is then performed once network connectivity is regained in order to record data with the analytics server.

Device Registration

In order to determine unique events, it is necessary to register each mobile device with the server. On launch of the app, the client automatically tries to register the device with the analytics server. If the device is successfully registered or the server responds that the device has already been registered, the client stores that information locally so as to avoid further unnecessary registration requests.

In order to ensure that data is truly anonymous, a unique TaleBlazer Analytics ID is generated from the device's ID. On iOS, this ID is installation specific and changes if the user reinstalls the app. On Android, this could be the device's actual device ID, depending on the version of the OS. To generate a unique ID for TaleBlazer Analytics,

the retrieved device ID is hashed with the SHA-256 cryptographic algorithm. SHA-256 is a "one-way" cryptographic function - the generated text cannot be decrypted back to the original text. This unique generated string is used by the analytics server to uniquely identify devices.

Session Request

In order to track gameplay sessions, a session has to be requested from the server at the start of a new game. On a successful request, the server responds with a unique session ID, which the client stores locally. The session ID is used to tag events to identify them as having taken place during a particular session. Sessions can only be requested once the device has been registered, as each session is tied to the device being used.

The TaleBlazer Analytics client also has to work in an offline capacity, ensuring that events are tracked and correctly recorded with the analytics server when network connectivity is reestablished. To this end, all sessions start off as "local sessions", each with a session ID unique to the device. Once the network becomes available, a "network session" is requested for each local session stored on the device. Local sessions are then converted to network sessions and any events tagged with the corresponding local session ID are updated to use the new network session ID.

Event Tracking

Once a session has been successfully established, events can be tracked and recorded with the server. Each event is tagged with the session ID it occurred in and the time the event took place, as well as event-specific information. Events are added to a queue and sent up periodically to the server in batches of configurable size. This minimizes the amount of network requests performed and the data sent.

Events are persisted locally in order to persist through application and device restarts, as well as to ensure that data is not lost in the face of spotty network connectivity. This allows event data from offline events to be kept and automatically sent to the server when network connectivity becomes available.

4.4 TaleBlazer Analytics Site

This section details the structure of the analytics site and its features. It also provides examples of the user interface.

4.4.1 Technical Overview

The TaleBlazer Analytics site allows users to view and download analytics data for their games. The front-end layout is accomplished using Bootstrap, a styling framework that provides a set of pre-styled elements, grid layout system, and extensible CSS classes. JavaScript is used to provide the behavior of the site, including the dynamic generation of UI elements, such as the tables for analytics data.

The site is built with a focus on client-side page rendering. All relevant analytics data is retrieved via AJAX calls to the Data Analysis API of the analytics server. A minimal amount of server-side HTML templating is performed to provide information such as when the game was first created and a list of its custom events, for filtering and navigation purposes. The site is also responsive - mobile and tablet devices can easily view information with a UI (user interface) suited to their device. Additionally, users can also download the raw set of analytics data.

4.4.2 Analytics Pages

Analytics pages are the pages of the site that provide the different types of analytics data for each game. The analytics pages of the site correspond directly to each API in the Data Analysis API group. In particular, these pages are the:

- Games Played Page
- Gameplay Duration Page
- Agent Bumps Page
- Custom Events Pages

All analytics pages share the same page loading behavior and common user interface elements. A typical analytics pages looks like Figure 4-1, which shows the user interface for the Games Played page. Appendix A contains additional figures of analytics pages.

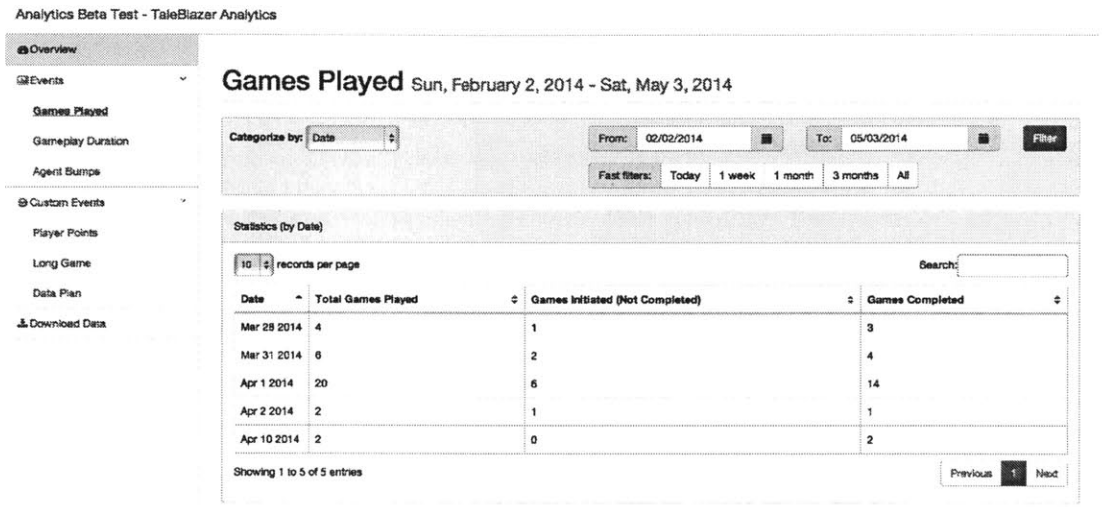


Figure 4-1: Games Played Page

Page Rendering and Loading

The pages of the site all correspond directly to each API in the Data Analysis API group. For example, the Agent Bumps Page gets its HTML and analytics information from the Agent Bumps API.

HTML requests to each API return a rendered HTML page with no embedded analytics data. On the server, the page is rendered by modularly composing a set of ECT templates. This results in a modular site and accomplishes maximum code reuse. The amount of server-side templating is minimal in order to ensure that the page is rendered and served to the client as fast as possible for short page load times.

When the page is fully loaded, an AJAX (Asynchronous JavaScript and XML) call is made to the same API, which responds with the corresponding analytics data.

The user interface of each page is then updated to reflect the new data.

The combination of fast page loads and AJAX calls for data ensures that the user is not stuck looking at a loading page while the analytics data is being calculated. It also eliminates page loads when filtering and categorization options are changed and new data is requested - the page's existing UI elements are simply updated when new data is received.

User Interface Elements

All analytics pages share common user interface elements, which not only maintain consistency throughout the site, but allow maximum reuse of code. This is a result of the modular construction of each page, as previously mentioned. The common user interface elements are the site navigation menu, the filter and categorization menu, and the data table.

Site Navigation Menu The Site Navigation Menu is a collapsible menu with links to all pages containing analytics information for a particular game. Notably, it includes a dynamic list of the custom events belonging to the game in question. Figure 4-2 shows the user interface for the menu.

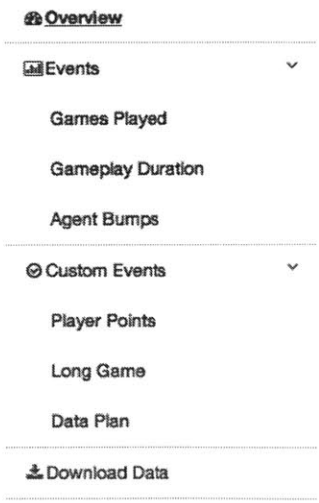


Figure 4-2: Side Navigation Menu

Filter Menu The Filter Menu provides all the options for filtering and categorizing analytics data for all analytics pages. Users can choose from a pre-defined list of categories to categorize their data by. They can also manually define the date range of data they would like to look at. Additionally, they can choose from a set of pre-defined date ranges, including the option to view all data for the lifetime of the game. The list of categories can be modified or the option completely disabled on a page-by-page basis, as it is on the Overview Dashboard (Figure 4-5). Figure 4-3 shows an example of the filter menu.

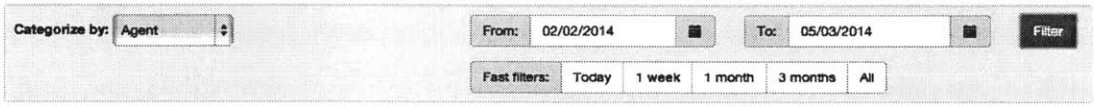


Figure 4-3: Filter Menu, with the categorization option enabled

Data Table Each analytics page displays its analytics data in the form of a data table. This data table is implemented using the DataTable JavaScript library. Analytics data tables are column-sortable and searchable across the entire set of results. Data tables paginate automatically based on the number of results. The user can optionally choose the number of results to view per data table page. Figure 4-4 shows what the data table for Agent Bump data, categorized by Agent looks like.

Overview Dashboard

The Overview Dashboard is a unique analytics page as it does not include a categorization option or a data table. Instead, the Overview Dashboard provides a concise set of statistics detailing the performance of the game in the form of a grid. As before, users can choose the date range of data to get statistics for. Figure 4-5 shows the UI for the Overview Dashboard.

Statistics (by Agent)

10 records per page

Search:

Agent ID	Agent name	Unique bumps	Total bumps
2	Friendly T-Rex	29	29
5	Stegosaurus	23	23
6	Brontosaurus	25	26
8	Apple	22	22
10	Happy Banana	22	22
12	Key	16	21
41	MIT Magic	1	1
44	Connection Question	22	23
56	Exit	11	11
60	Inventory	2	3

Showing 1 to 10 of 10 entries

Previous 1 Next

Figure 4-4: Data table for the Agent Bumps page, categorized by Agents

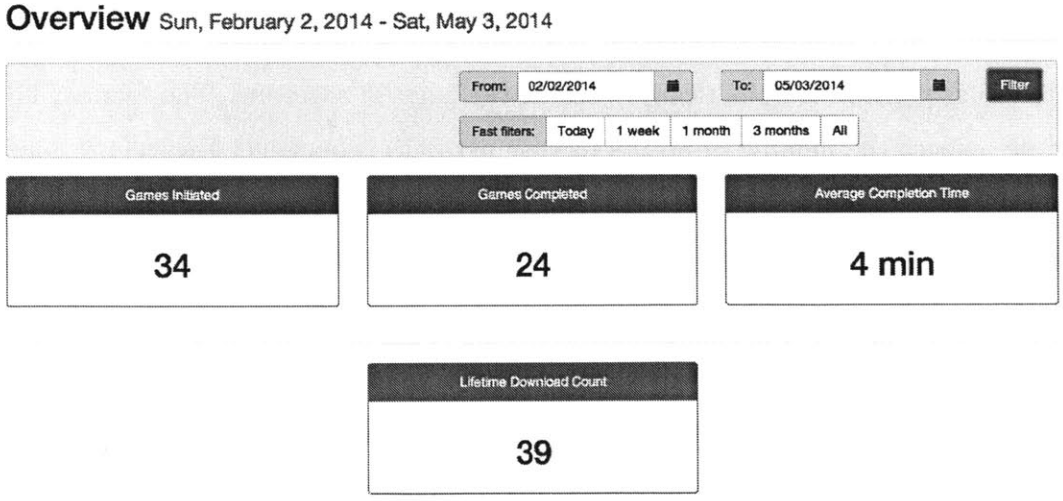


Figure 4-5: Overview Dashboard

4.4.3 Data Download

The analytics site also allows users to perform their own data analysis on the collected data. Users can download a CSV (Comma-Separated Value) file containing the set of all events that occurred over a particular time period. In particular, this file contains all Agent Bump, Region Switch, and Custom Events that occurred, as well as information about the session that they occurred in and the version of the game that was played.

This functionality allows users to not only perform data analysis on the events of a particular game, but also gives them the ability to view user behavior across separate games using the unique TaleBlazer Analytics device ID.

Chapter 5

Testing

This chapter details the testing that was performed on the TaleBlazer Analytics system. It also gives the results of the tests and their consequences on the development process.

5.1 Overview

The TaleBlazer Analytics system underwent a series of internal and external tests in order to test the robustness and accuracy of the data collection system. Two types of tests were performed: internal alpha tests and external beta tests.

Alpha tests were performed internally by the TaleBlazer team and focused on testing the performance of the client under adverse network connectivity conditions. They also focused on making sure that data was being collected properly. A beta test was performed with our partner institutions to test the system in a production capacity, as well as serving as an additional data collection test. Finally, a user interface test was performed with our partner institutions to get their feedback on the final analytics site.

5.2 Internal Alpha Tests

Multiple and extensive alpha tests were performed on the TaleBlazer Analytics system primarily to identify bugs and issues with the interaction between the analytics server and client. The main goal of these tests was to identify edge cases in the client's API workflow and to ensure that data was being persisted and sent to the server correctly.

A test TaleBlazer game was created that included every different type of possible interaction that could result in a trackable event. The game was tested by multiple members of the TaleBlazer team in varying network conditions on Android and iOS devices. The Android devices consisted of HTC Droid Incredible phones running Android 2.3.3, with and without an SD card present. The iOS devices included iPhone 4 and iPhone 5 phones, running iOS 6 and 7.

Each test lasted about 30 minutes. Testers were asked to play the test game simultaneously as they would normally play a TaleBlazer game. Each tester was asked to write down the time and order in which agents were bumped and actions were taken. This information was then compared against the data analytics that were captured.

5.2.1 Effects on Development

The alpha tests allowed us to identify multiple edge cases in the client's API workflow that resulted in malformed data to be sent to the server. This allowed us to harden the client's workflow to make it as robust as possible in the face of these unforeseen edge cases.

Alpha testing also alerted us to the need to track local sessions and events. As a result, the session request mechanism for the client was modified to implement the concepts of local and network sessions (see 4.3.2). This change not only gave us the ability to track local sessions, but it also made the client more reliable in the face of network connectivity issues.

5.3 External Beta Test

After the alpha tests had been concluded, a beta test was performed primarily to test the performance of the system in a production capacity. Partner institutions were asked to participate and provide feedback on their experience. The test TaleBlazer game from the internal tests was modified to not only serve as a test for data collection, but also to introduce our partners to the way that the analytics system worked. Partners were asked to provide written logs of their in-game actions to allow us to verify the accuracy of the collected data.

5.3.1 Effects on Development

The beta test allowed us to verify that data was being collected correctly and that the client was performing well on multiple types of Android and iOS devices. The most important result of the beta test was that it alerted us to issues with the deployment of the analytics server.

For the tests, the analytics server was deployed to a port that was not port 80, typically used for HTTP connections. One tester ran into an issue where the server was not receiving any data from the testing devices. The logs showed that no connections were being received from those devices. We were then able to determine that the network in question restricted HTTP requests to port 80. As a result, this drove the adoption and deployment of Nginx as a proxy server, in order to allow connections to the analytics system from restrictive networks.

5.4 Final User Interface Test

After the external beta test had concluded, partners were directed to the final deployed version of the analytics site. Partners were provided with minimal instruction on how to use the site, in order to determine the site's usability and what elements were unclear to new users. In addition to requesting general feedback, partners were asked the following questions:

- Is there any data you are interested in that is not currently being collected?
- Is the graphical user interface (GUI) intuitive?
- Is the data displayed in a useful way?
- Are there additional views of the data that would be useful to you?
- Are there misleading labels or elements on the site?
- Is the downloaded raw analytics data in a file format useful to you?

5.4.1 Results

In terms of the data, partners felt that the data being collected was indeed useful to their purposes, which was in line with their feedback on the original mockups. Particularly, the partners were interested in the Overview page and the Games Played page. They felt that this information provided them with useful statistics on the popularity and status of their game. They also liked the ability to download the raw analytics data, which allowed them to calculate their own statistics.

One of the most requested features was the addition of visualizations in order to make the data simpler to understand. Additionally, partners felt that help or tutorial elements be added in order to explain certain data labels and functions of the site. Interestingly, some partners requested features that were implemented in the custom events system, indicating that an FAQ or help section would be incredibly useful in on-boarding new users. Overall, the test indicated that partners found the site useful and that the user interface could use minor tweaks in order to make it simpler to use, specifically emphasizing the creation of help or tutorial pages.

Chapter 6

Future Work

This chapter gives recommendations for the future of the TaleBlazer Analytics system, including new features and technical improvements.

6.1 Data Visualizations

Currently, the analytics site only provides a tabular view for analytics data. Although this data provides useful information for analytics users, it can be difficult to get a high-level understanding of the statistics quickly. As a result, it would be useful to implement visualizations of the analytics data.

Data visualizations would allow users to quickly interpret and draw conclusions from analytics data. This feature would improve the usability of the site and allow users to focus more on the overall results of the data, rather than having to interpret the results from the tabular data. For example, Figure 6-1 shows a visualization detailing how many players played specific versions of a game over a week.

The TaleBlazer Analytics system was built to accommodate visualizations from the start of its development. The relevant data is already provided in JSON via the same API call that allows pages to retrieve their relevant analytics data. Visualization implementation is easily accomplished via D3.js, a popular JavaScript visualization library.

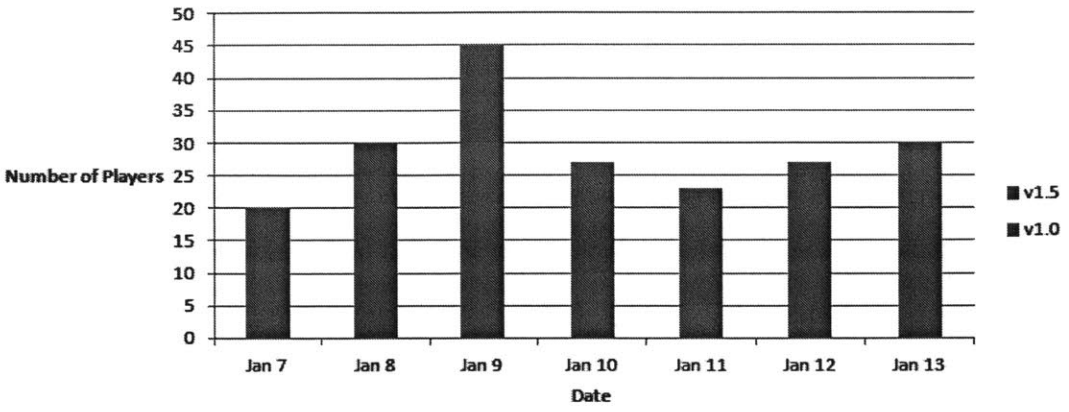


Figure 6-1: Distribution of Total Players by Game Version, over a week

6.2 Authentication/Authorization

In order to restrict users to view only data for their games, it is necessary to implement an authentication and authorization system. Although a simple login and permission system could be set up for TaleBlazer Analytics quickly, it is important to build a system that allows users to use their same credentials from TaleBlazer Server. This would avoid the problem of having multiple credentials for the two services. This system was delayed in order to prioritize the data collection and analysis features of TaleBlazer Analytics.

A way to implement this would be for the analytics server to utilize the same session information currently used by TaleBlazer Server. A simple solution would be for TaleBlazer Server to use database-backed sessions, which would allow the analytics server to access the session data directly.

A more extensive solution would be to implement OAuth2, which is a protocol for security authentication and authorization. This solution would involve implementing OAuth into TaleBlazer Server or deploying a separate server solely responsible for authentication and authorization. Plugins for CakePHP currently exist for enabling this functionality.

6.3 Improved Statistic Calculation

Statistics are currently calculated every time that a request to an analytics page is made. Although the statistic calculations are optimized to offload the majority of the work to the database, the calculation times may take longer as more data is collected. Two options for mitigating this issue are to perform progressive calculation and to store the calculation results using an in-memory cache.

Progressive calculation means that statistics are calculated and stored separately as the data is received from TaleBlazer Mobile. As a result, statistics are already available when requested by a page and so the computational overhead per request is significantly decreased.

An in-memory cache would solve the issue of unnecessary recalculation. In-memory caches, such as Redis and Memcached, allow incredibly fast access to data because the data is stored in RAM. Cached data can be expired and removed based on conditions set by the developer, such as newly arrived data. Storing the results of calculations in an in-memory cache would solve the issue of unnecessary recalculations. A typical request would consult with the cache first to see if the statistics are already stored. If still valid, the request would return the statistics. Otherwise, the server would perform the calculation as normal.

6.4 Code Improvements

Future technical improvements to the server codebase involve creating a Service layer to consolidate common model-related functions and the use of the `async` JavaScript library for improving the readability of asynchronous code.

Currently, the controllers for each API contain code to modify, retrieve, and save models as required by the API. A Service layer would remove model-specific code from controllers and place them into modules known as Services. Each service would be responsible for handling common functions related to a model. For example, a Device Service would handle common device registration tasks and duplicate device

checks. Ultimately, this would allow controllers to focus on validating inputs and responding to requests. The majority of the work with models would then live in the corresponding service.

The `async` JavaScript library provides a set of utility functions which handle common asynchronous tasks. The asynchronous nature of JavaScript code results in multiple levels of callback functions, which is often difficult to read and modify. The `async` library solves this issue by providing a powerful set of utility functions that improve the readability of the code. Compare the readability of the two functions in Listing 6.1, which perform the same tasks.

```
1  // Nested callbacks
2  // Each function waits on the results of the previous one before
       continuing
3  performAction1(function(){
4    performAction2(function() {
5      performAction3(function() {
6        // and so on...
7        })
8      })
9    })
10
11 // Callbacks using async library
12 // Results of each function are passed to the other using the next
       parameter
13 // When all are executed, the finalCallback is executed
14 async.waterfall([
15     performAction1(next),
16     performAction2(next),
17     performAction3(next)
18   ], finalCallback)
19 });
```

Listing 6.1: Comparison between normal callback code and `async` code

Chapter 7

Contributions and Conclusion

7.1 Contributions

Working closely with the TaleBlazer developer team, the following contributions have been made. First, key gameplay metrics were identified that would provide useful information regarding the performance of TaleBlazer games. Second, the TaleBlazer Analytics system was developed: a maintainable, scalable, and extensible system for the collection of gameplay metrics and the analysis and presentation of useful analytics data. Third, future improvements to TaleBlazer Analytics were identified and the groundwork laid in preparation for future development.

7.2 Conclusion

TaleBlazer Analytics is an automated data collection system that seamlessly integrates with the existing TaleBlazer platform. The system is composed of an analytics server, client, and site. The analytics server is responsible for receiving, storing, and analyzing gameplay data. The analytics client integrates with TaleBlazer mobile to collect and send data to the analytics server. The analytics site provides useful and comprehensive analytics data for TaleBlazer games. The system is built to be maintainable, scalable, and extensible so as to accommodate future development and expansion. TaleBlazer Analytics will help game designers and researchers gain

a better understanding of how players play TaleBlazer games and their educational impact.

Appendix A

Figures

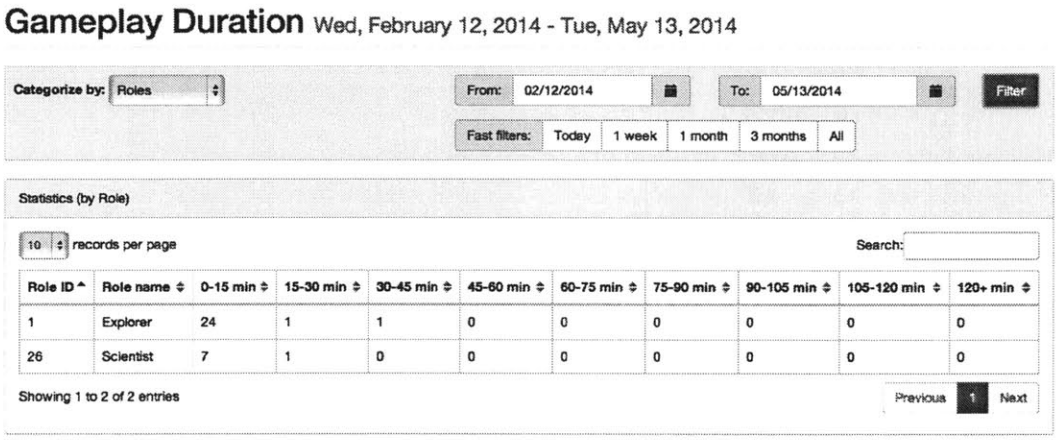


Figure A-1: Gameplay Duration, categorized by Role

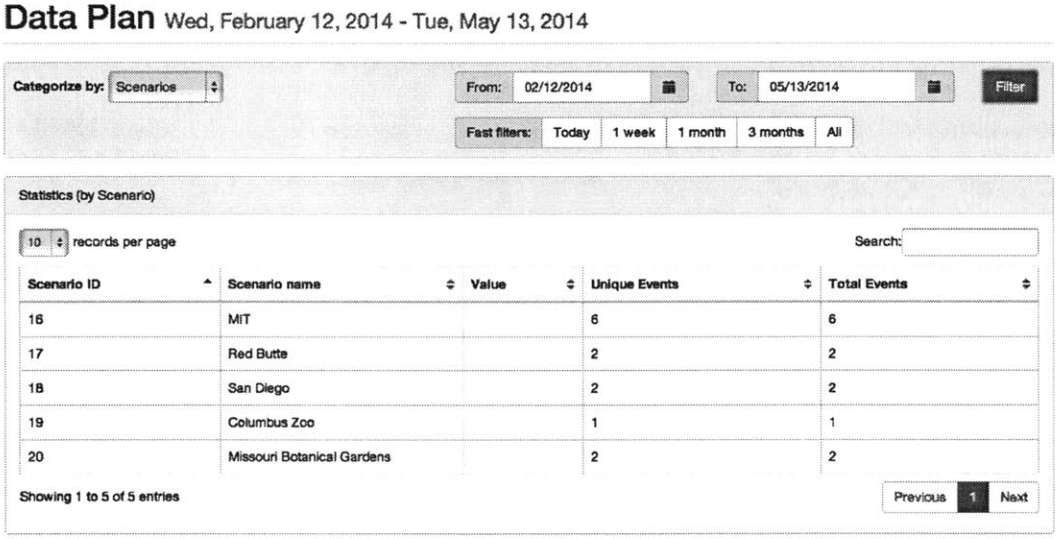


Figure A-2: Custom Events page, categorized by scenario. A "Data Plan" event is shown, which was captured if a user indicated in game if the device they were using had a data plan.

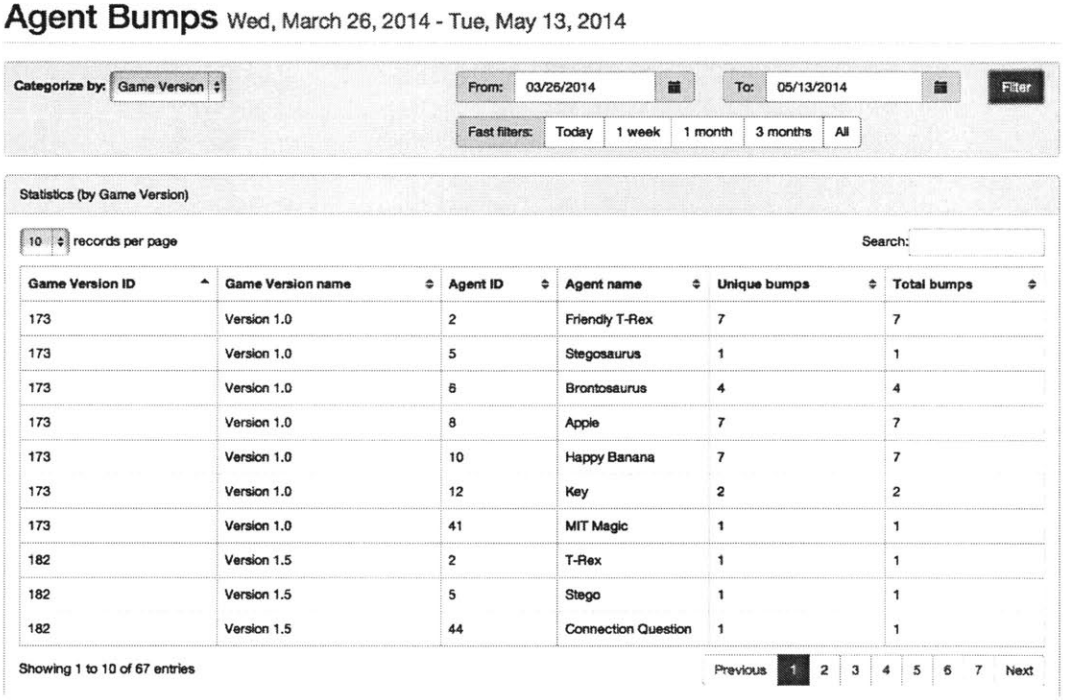


Figure A-3: Agent Bumps page, categorized by game version.

Bibliography

[1] Flurry Inc. Privacy Policy. http://www.flurry.com/legal-privacy/privacy-policy.

[2] OmniTI Labs. JSend specification. http://labs.omniti.com/labs/jsend.

[3] Sarah E Lehmann. Taleblazer: Implementing a Multiplayer Server for Location-Based Augmented Reality Games. Master's thesis, Massachusetts Institute of Technology, September 2013.

[4] Michael Paul Medlock-Walton. Taleblazer: A Platform for Creating Multiplayer Location Based Games. Master's thesis, Massachusetts Institute of Technology, June 2012.

[5] MIT Scheller Teacher Education Program. Environmental Detectives. http://education.mit.edu/ar/ed.html.

[6] MIT Scheller Teacher Education Program. Starlogo TNG. http://education.mit.edu/projects/starlogo-tng.

[7] Alex Rodriguez. RESTful Web services: The basics. https://www.ibm.com/developerworks/webservices/library/ws-restful, November 2008. DeveloperWorks, IBM.

[8] Mikito Takada. Understanding the node.js event loop. http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/, February 2011.