TRUE Instrumentation: Tracking Real-Time User Experience in Games

Eric Schuh, Daniel V. Gunn*, Bruce Phillips*, Randy J. Pagulayan*, Jun H. Kim, and Dennis Wixon*

Microsoft Game Studios

Eric Schuh has been interested in games ever since a square white ball bleeped and blooped across his screen in the early 1970s. He has been a member of the Games User Research team since 2002. In that time, he’s worked on franchises such as Fable, Project Gotham Racing, Forza, and Crackdown, developing innovative research techniques to answer complex research questions. Eric has a long history doing user research on how people use technology dating back to 1993, working on projects as varied as MSN, Access, printers, and network administration tools. Not a bad career for a guy who almost, but not quite, got his PhD from the University of Washington in social psychology.

*For photos and biographical information about these authors, see Chapter 4’s introduction.
Jun Kim is a User Research Engineer for Microsoft Game Studios and has been a member of the User Research group since 2000. He has worked on numerous games in the group and is currently working on creating innovative tools, data analysis, and methods to collect user behavior and attitudinal data. Before joining Microsoft, Jun received three years of postgraduate study in developmental and neurobiology at the University of Washington, studying primate models of human developmental disabilities.

Objectives in the Chapter

The present chapter describes a robust instrumentation solution that the Games User Research group at Microsoft has developed to improve user experience in games. The objectives of this chapter are to walk the reader through the development of our TRUE instrumentation, describe some case studies in which it was used successfully at various stages of the development cycle to improve games, and leave the reader with our lessons learned over the course of its development. It is our hope that the reader can take this information and successfully implement TRUE instrumentation in their games. We hope that not only will individual titles benefit from using our approach but that user experience in games overall will be improved with wider adoption of TRUE instrumentation.

Abstract

Collecting customer feedback on how people play your game—identifying where the design is too confusing, too easy, too lethal, and so on—can dramatically improve the users' experience with the game and increase its chances of success in the marketplace. Although there are many methods for collecting users' impressions and experience of games—focus groups, usability testing, playtesting—feedback from these is based on a limited exposure to the game. When these methods are used, there is a pragmatic reason for limiting the feedback from users: it is labor intensive to observe people playing the game at a level of detail that is needed to spot problems and identify their underlying causes.

This chapter outlines a system for the automated collection of gameplay feedback, enabling development teams to understand how people experience the entire length of the game. By having the game automatically log behaviors of interest—player deaths, items collected, levels completed—it is possible to collect large amounts of data efficiently. When paired with other data streams, such as captured video and in-game surveys, it is possible to understand what people are doing in your game, what elements of the design are causing them to behave the way they
do, and how they feel about their experience of the game. In many cases, understanding behavior, its causes, and user evaluation is precisely the information you need in order to improve your game.

Automated capture of user data is not new. In games, post-match stats and leaderboards have been around as long as videogames have. Although relatively rare, using these data to improve a game is not unique: Valve has used automated data capture and Steam to tweak the difficulty of *Half Life 2: Episode 2* after it was released. This chapter will provide details on how to use automated data collection to improve a game before it is released, improving the quality without negatively impacting the schedule. We will illustrate how Microsoft Game Studios has used automated data collection to improve games at various stages of development. In addition, we will share best practices we developed in the course of using this form of data collection for the past five years on over thirty games.

### 15.1 The Genesis

#### 15.1.1 *Voodoo Vince*

In September 2003, Microsoft Game Studios (MGS) released a little platformer developed by Beep Industries, a title by the name of *Voodoo Vince*. The titular hero is a voodoo doll who must jump, collect, jetboat race, ride rats, lure zombies, self-immolate, and battle his evil twin in an attempt to rescue Madam Charmaine from the clutches of Kosmo the Inscrutable. Besides a quirky hero and infectious score, *Voodoo Vince* also marked the first time MGS utilized instrumentation to collect consumer feedback about play throughout an entire game.

#### 15.1.2 The Problem

The problem we were trying to address was fairly straightforward—we wanted to understand what issues people encountered late in the game, why they were having those difficulties, and have a good idea of what we needed to do to fix those problems.

We contemplated using some of the standard user research techniques we typically employ at Microsoft Game Studios. We considered doing traditional think-aloud *usability testing*, where we bring target gamers in and observe them playing the game and doing specific game-related tasks (See Figure 15.1) (Dumas & Redish, 1999; Fulton, 2002; Medlock, Wixon, McGee, & Welsh, 2005; Nielsen, 1993; Medlock, Wixon, Terrano, Romero, & Fulton, 2002; Pagulayan, Keeker, Fuller, Wixon, & Romero, 2007; Pagulayan, Steury, Fulton, & Romero, 2003). We felt that we couldn’t afford the ~160 hours of observation time that testing would cost. The results would be irrelevant by the time it was finished because the design of the game would have changed by the time the results were analyzed.
Another option is Playtesting. Although a common term in the game industry, playtesting means different things to different teams. At Microsoft Game Studios, Playtesting refers to a structured method for collecting consumer feedback via a survey about different aspects of the gameplay experience (Pagulayan et al., 2007). Typically, participants play the game for a set period of time and then fill out a questionnaire asking them to rate different aspects of the game, such as fun, graphics, goal clarity, frustration, and so on. Because the data collected in Playtest are survey responses (and not observations of what people do), it is possible to collect data from multiple participants simultaneously. Our Playtest labs at Microsoft Game Studios support data collection from up to fifty-one gamers at one time. Figure 15.2 shows one of the three Playtest labs.

The fact that data can be collected from people simultaneously in playtest helps bypass the time-consuming nature of usability testing—it is be possible to collect feedback on 20 hours of gameplay from fifty-one people in a long weekend spent in the playtest labs. However, the tradeoff is granularity—you detect big things that are wrong (say, a common perception that aiming was too difficult), but not more nuanced problems (say, vertical aiming with the sniper rifle being too difficult when trying to headshot a Brute). You simply don’t have the fidelity of data from broad survey responses to detect this level of interaction—and problems often lurk at this layer of complexity.

By 2003, we had completed a lot of usability testing to improve the core mechanics of Voodoo Vince (platforming, combat, powered attacks, puzzle solving) and the initial experience. We conducted Playtesting to determine the broad satisfaction of things like character movement, combat effectiveness, and double jump responsiveness, and funneled that feedback back into the game design. But the team wanted
more information. Were the platforming sections in the carnival portion of the world (the last level) too difficult? Where were people missing jumps? What puzzles were people failing to complete throughout the game, and why? How long did it take to beat the bosses, and how many attempts did it take?

15.1.3 The Solution

In order to attempt to answer these and other questions, we built a small application for users to log their basic behaviors (for example, click a button to indicate that you started a level, click a different button to log a death, another one to indicate confusion with a puzzle, and so on). We deployed this in our Playtest labs, and had a group of seventeen people log their own activities and perceptions as they played the game for 8 hours. The resulting data were noisy—some people forgot to use the logging application as the session went on, others double-clicked rather than single clicking (inflating frequency counts). Nonetheless, the data were promising. We had a rough idea of where people were dying and why, how long it was taking to complete a level, and which puzzles were most confusing.

Our next step was to take the noise out of the system. Rather than having users log every time they died, we instrumented the game, setting hooks into the code itself to automatically record each death and write it to a logfile. This allowed the user to concentrate on playing the game rather than logging data, producing more accurate, detailed, and actionable data.

We then had seventeen participants play the instrumented version of the game in our Playtest lab one late summer Saturday. On reviewing the resulting data,
we noticed people spending an inordinately long time and having lots of falling deaths on Chapter 19 (of 32)—*The Rat Rodeo*. This level is a Boss Battle involving a screeching opera statue (the boss), a crumbling floor, falling bricks, and our hero Vince riding a flea-bitten rat as shown in Figure 15.3. The player must guide the rat around a circular room, maneuvering so that falling bricks hit Vince on the head (he is a Voodoo Doll, after all, so hurting him hurts his enemies) while jumping over pits that open up as the floor disintegrates. Making matters worse, the enraged statue periodically shrieks, sending out a shock wave that pushes poor Vince into a waiting pit.

The chapter proved very difficult—it was hard to tell where the falling bricks would land, so it was tricky to maneuver the rat into the proper position; the floor would disintegrate with little warning, sending players to their doom; and it was difficult to judge where the shock wave was, making it almost impossible to avoid. As a result, people were dying left and right, were spending over an hour in the chapter, and were generally frustrated. See Figure 15.4 for a chart of deaths in this portion of the game.
15.1.4 Design Impact

Based on these data, we made several changes designed to reduce the deaths and time spent while still retaining the challenge needed for this to qualify as a boss battle. We added shadows to the bricks, making it easier to see where they would land and providing a spot for players to aim for while steering their rat. The disintegrating floor got a makeover, wherein soon-to-disintegrate pieces got a warning animation and visual highlight that helped users identify those pieces early enough to avoid them. Finally, a red glow was added to the statue’s sonic blast, making it easier to see and therefore easier to jump. These changes combined to make a chapter that had previously been nearly impassable beatable after a few attempts—a fitting challenge for the completion of this portion of the game.

15.1.5 Our Instrumentation Refinement

Although Voodoo Vince didn’t find success in the marketplace (it was a platformer on a console that skewed heavily toward first-person shooters, sports, and racing games), the game had lasting impact. The basic idea of instrumenting the game to collect extended gameplay feedback from consumers was deemed a great success so development effort was set aside to build a tool that could be used by all games to collect this information.
We developed a series of requirements for this tool which formed the foundation for what we have called the TRUE (Tracking Real-Time User Experience) method. The hallmarks of TRUE instrumentation include:

1. Collection of attitudinal feedback via in-game surveys.
2. Collection of contextual data as well as the main data or interest
3. Utilizing captured video to better understand the data

15.1.6 Surveys

When specifying the requirements for the TRUE instrumentation tool, we looked at the shortcomings of the data collected in Voodoo Vince. One of the biggest flaws was that we only captured behaviors and not any attitudes. Recording what people do is vitally important, but it only tells an incomplete story. It captures the mechanics and dynamics of a game but not its aesthetics (Hunicke, Leblanc, & Zubek, 2001). Knowing that someone died ten times against a particular enemy is interesting but hard to interpret. Is the person frustrated by the repeated deaths at the hand of the same enemy, or are they enjoying the challenge associated with figuring out how to take out an effective adversary? Put differently, are these ten deaths a problem that need to be addressed, or are they a key component to the overall enjoyment of the game that should be preserved? Without collecting the attitudinal data, we will never know.

To address this shortcoming, we added the ability to include brief surveys within a game itself. At certain points of a game, the game would pause and display brief questions on the screen as seen in Figure 15.5. The participant would use the controller to select a desired response, and then hit the A button to register that feedback. There were 3 main categories of in-game surveys we wanted to support, each suited for answering different sorts of questions. These categories are:

- Event-based surveys—surveys that are displayed when a certain event occurs, such as completion of a mission, player death, or solving a puzzle. This type of survey is useful for getting basic feedback about the experience that led up to the event (e.g., "How difficult was that mission?").

- On-demand surveys—surveys that the user can bring up whenever they want to provide feedback. This allows people to spontaneously indicate whenever they feel lost, bored, confused, or happy.

- Time-based surveys—surveys that appear after a set interval of time. These surveys are particularly useful for assessing satisfaction with progression or mapping out enjoyment over time. One danger of using time based surveys is making sure that they do not negatively intrude on the gameplay experience itself. If the participant is interrupted every minute asking how much fun they are having,
eventually they will become frustrated with this constant intrusion. To avoid this problem, we recommend that you display surveys as *infrequently* as needed to get the information you need (we have ranged from every 3 minutes to every 10 minutes), and that they are displayed when there is a natural break in the action (after combat is complete, not in the middle of a sword swing).

### 15.1.7 Contextual Data

Another important factor for collecting effective survey data—and this applies to the behavioral data as well—is to provide the relevant *contextual information* needed to interpret the result. Don’t just record that a participant indicated the game was “Not fun”—indicate the timestamp of the survey response, what level of the game the player was on, whether they had just succeeded or failed the relevant quest, and so on. In order for information to be actionable, you need to understand the context in which it was collected. The specific contextual information will vary based on the details of the genre and game being tested, but there is some information that we always collect with every data point (both behaviors and survey responses). These are:

- **Build number**—ideally you are going to be testing the game several times, so you are going to want to know which build a particular data set is related to.
- **Test name**—you may have different studies going on with the same build, perhaps a test of novice vs. expert players and want to retain the ability to
separate or combine the data from the two different groups. Having a test name field recorded with every data point allows you to do so.

- Participant ID—you want to be able to identify which participant the data are coming from.

- Timestamp—you need to know when the data were collected.

- Difficulty setting—you want to be able to tease apart whether a problem you identify in the data is common for everyone, or just people playing a certain level of difficulty.

- Chapter name/mission name/quest name/level name/map name—the specifics will depend on what type of game you’re testing, but there should be some indication of what portion of the game the data come from.

- Position coordinates—recording the $x$, $y$, $z$ coordinates of every piece of data allows you to display that information on a map, which is an incredibly powerful way of identifying where problems are occurring. See example in Figure 15.6.
15.8 Video

We also added captured video to our instrumentation so that while participants play the game, a video capture card records their on-screen activity. These videos are then synched with the instrumentation data using the always-present timestamp information. The combined data can then be included in a SQL database, upon which detailed reports can be built (see Figure 15.7). The resulting reports allow researchers and team members to skip painful hours of reviewing videos, instead jumping directly to the item of interest (a death, a level completion, a survey response of “I’m lost”, etc.).

15.2 Putting it Together: TRUE Instrumentation

Coming out a little over a year after Voodoo Vince, Halo 2 made extensive use of our TRUE instrumentation solution. We were interested in smoothing out the rough spots of the single player campaign, so we captured data from nearly 400 participants over approximately seven weeks using 13 different builds in the summer of 2004, covering each mission anywhere from three to eight times. Such a feat would not be possible without instrumentation.

15.2.1 A Halo 2 Example

Halo 2 is a first-person shooter video game released in the fall of 2004. There are twelve single-player missions that take approximately ten to twelve hours to complete.
**Figure 15.8**

Mean number of player deaths over each mission in Halo 2.

The missions are linear, containing between ten and thirty smaller encounters. Overall, there are over two hundred encounters in the Halo 2 single-player campaign.

To implement the TRUE method, we partnered with designers and engineers and automatically recorded player deaths, which enemy killed them, what weapon the enemy was using, and other combat related variables. We also recorded the relevant contextual information with each data point—what mission, which encounter, and what time the event occurred. For the attitudinal data, we used a time-based in-game survey (approximately three minute intervals) asking players to rate their perception of the game’s difficulty and their sense of progress through the game. Finally, we captured video as each participant played the game and synced that up with our instrumentation data.

On average, a given testing session consisted of approximately 25 participants that came onsite to play through the campaign over the course of two days. After a given session, we were able to quickly view participant performance at a high level across the individual missions (see Figure 15.8). In this example, we were interested in the number of times a player died in each mission.

As shown in Figure 15.8, there were more player deaths in Mission 10 (M10) than in the other missions—more than we had expected. However, viewing total deaths across missions did not tell us how participants died or whether participants found this frustrating.

To better understand what was happening, we drilled down into the specific mission data to see how participants died in each of the encounters comprising this mission (see Figure 15.9). Using a Web front end, we could simply click on the bar of interest in the graph to drill down to another level of detail to the average deaths for each encounter within Mission 10. From there, we observed a potential problem in the third encounter of the mission, as you can see clearly in Figure 15.9.
Although these data helped us locate the area of a potential issue with the mission difficulty, it did not provide sufficient information to explain what, in particular, was causing participants difficulty. We knew that during this particular encounter in the mission, participants were fighting successive waves of enemies in a large room. However, we did not know what specifically was causing them to die.

To figure this out, we drilled down even further into specific details of this encounter. Specifically, we were able to break out deaths into the particular causes. In this example, the Brutes (one of the enemies the player had to defeat) were responsible for 85 percent of participant deaths.

Drilling down into the data even further (again, by the click of the graph), we identified three primary ways participants were dying: Brute Melee attacks, Plasma Grenade Attach (the grenade sticks to the player), and Plasma Grenade Explosions. Being able to isolate the exact cause of deaths was important because in Halo 2 there are numerous ways enemies can kill a player. This successive drill down approach allowed us to quickly discover the main causes of participant deaths within minutes.

However, we still did not completely understand how this was killing the participants. Because the combat system in Halo 2 is complex, we turned to the designers of that system to provide further insight and also took advantage of a key component of our instrumentation—captured video. For each death we could link to a video that showed us exactly what happened. With the game designers by our side, we viewed several of the participant deaths—in particular, deaths where participants died due to direct hits from a plasma grenade.

After watching these videos, the designers were able to immediately pick up on a subtle nuance in the game mechanic that only they were able to identify. The Brutes in this section of the game threw grenades faster and with less of an arc (compared
Number of player deaths before and after design changes were implemented.

to other enemies which could throw plasma grenades) which gave participants less time to react.

Using this information, the designers made several changes to reduce difficulty. Specifically, they prevented Brutes from throwing Plasma Grenades altogether (for that encounter). In addition, they reduced the overall number of enemy waves and spawned enemies in only one location. Previously, enemies could spawn from several locations in the room players were fighting in. This minimized the chance that players would get melee’d from behind.

A week later, we tested a new build of the game that included the fixes the designers had generated. We brought in a different set of participants, had them play the game, and then checked to see whether the design changes worked. In looking at the new data, we saw a dramatic reduction in the number of deaths, especially those from Brute Melee’s and by plasma grenades (See Figure 15.10).

So, we solved the problem of too much death. But did we go too far? In tweaking the Brute behaviors (who are supposed to be scary and lethal) and the way plasma grenades were to be used (which are supposed to be devastating), did we make this encounter too easy? We turned to the in-game survey data to answer that question. Similar to the way we drilled into the death data, we were able to quickly assess the frequency of responses for the time-based survey data. As Figure 15.11 shows, the changes did not negatively affect how people perceived this encounter. The percentage of responses indicating that the level of difficulty was “about right” jumped
Percent of player responses before and after design changes were made.

from 43 to 74 percent, while the percentage of people who felt the encounter was “too easy” stayed incredibly flat and very low (only 3 percent indicated it was too easy after making the changes).

As a reminder, the real-world example we’ve been discussing pertains to only 1 of the 211 encounters in Halo 2. We discovered many issues across the entire campaign, worked with designers on changes, and verified that those design changes worked. This would not have been possible without TRUE instrumentation as well as the experiences gleaned from doing this sort of research on a variety of games in various points of the development cycle.

The following sections provide examples of what data collection and impact can look like at various points in the game development lifecycle. We start off with a simple, straightforward, example of assessing design intent during the polishing phase of production utilizing TRUE. Next, we discuss the utility of TRUE testing during the beta phase of a project. Finally, we demonstrate how TRUE can be successfully utilized to inform demo construction of a game.

15.3 Forza 2: Production and Polishing

The bulk of the data collection will be done at the end of the production phase as the builds stabilize and polishing occurs. During the latter part of this time period, any user research that gets done needs to be done at an extremely fast pace. Features are being tweaked and locked down much more quickly than earlier in the product lifecycle, and this frantic pace mandates that any user feedback coming in needs to be in step with production—or it cannot be responded to. When instrumentation is done right, it allows us to keep in sync with the feverish pace of late production and to maintain the feedback loop between end user and the production
team right up until ship. We conducted one particular TRUE test on Forza 2 during late production which demonstrates how valuable data can be gleaned very quickly, very late in production, and still have a tremendous impact on a game.

Forza 2 is a racing simulation game released in the Spring of 2007. One defining characteristic of the game is its realistic driving physics; thus, trying to drive a Ferrari F430 GT is considerably more challenging than driving a Ford Focus (as it would be in the real-world). The game includes two single player modes, Career and Arcade, in addition to a multiplayer mode. In Career Mode, players start off with access to slow, low-end, cars and gradually earn credits and access to faster cars by doing well over multiple races. In Arcade Mode, players have the ability to jump right in and race faster high-end, and typically more difficult to handle, cars. While we used TRUE to help balance and tune progression in Career Mode during production, there was one big aspect in Arcade Mode that we wanted to get user feedback on before the game shipped, and that was the Time Trials. Yet, our window of opportunity for testing the time trials in Arcade Mode came at a point very late in production. In fact, we had only one week to run the test, turn the data around, and make changes to the time trials before they were locked down permanently. The following example relates to our efforts to polish the Arcade Mode time trials.

Forza 2 contains 25 different time trials in Arcade Mode. The player’s goal for each trial is simple: Complete at least one lap on the track faster than the pre-specified target lap time for that trial. If the player successfully beats the target lap time, the car used for that trial is unlocked and added to the player’s Arcade Garage. If the player is not successful, the car remains locked and players cannot use it in other parts of Arcade Mode. One challenging aspect for players is that the car, the track ribbon, and the target lap time to beat are all preset. Thus, if players would typically shy away from racing with the powerhouse Nissan Silvia Top Secret (a car not available until well into Career Mode) because of its notoriously intractable handling, in order to beat time trial #3, they would have to use that Silvia. In addition, it wouldn’t be the Silvia on an oval track, it would be on a challenging real-world race track (i.e., Tsukuba) and the lap time would need to be under 46 seconds to boot!

While the design intent behind the time trials was that they be challenging for players, the designers did not want them to be overly frustrating. Indeed, what could be more frustrating to a player than the inability to unlock a car for their arcade garage because of a target lap time they perceive to be impossible beat after the 50th unsuccessful lap? Luckily, the Forza 2 designers had a clear design intent for the time trials that we could use to test against: The time trials should be challenging to players but approximately 80 percent of the target users should be able to complete any particular time trial and unlock the car after ten laps.

15.3.1 The Problem

At the point in production that the twenty-five time trials were finally getting locked down, we still did not know whether the design vision for them was being realized.
team right up until ship. We conducted one particular TRUE test on Forza 2 during late production which demonstrates how valuable data can be gleaned very quickly, very late in production, and still have a tremendous impact on a game.

Forza 2 is a racing simulation game released in the Spring of 2007. One defining characteristic of the game is its realistic driving physics; thus, trying to drive a Ferrari F430 GT is considerably more challenging than driving a Ford Focus (as it would be in the real-world). The game includes two single player modes, Career and Arcade, in addition to a multiplayer mode. In Career Mode, players start off with access to slow, low-end, cars and gradually earn credits and access to faster cars by doing well over multiple races. In Arcade Mode, players have the ability to jump right in and race faster high-end, and typically more difficult to handle, cars. While we used TRUE to help balance and tune progression in Career Mode during production, there was one big aspect in Arcade Mode that we wanted to get user feedback on before the game shipped, and that was the Time Trials. Yet, our window of opportunity for testing the time trials in Arcade Mode came at a point very late in production. In fact, we had only one week to run the test, turn the data around, and make changes to the time trials before they were locked down permanently. The following example relates to our efforts to polish the Arcade Mode time trials.

Forza 2 contains 25 different time trials in Arcade Mode. The player’s goal for each trial is simple: Complete at least one lap on the track faster than the pre-specified target lap time for that trial. If the player successfully beats the target lap time, the car used for that trial is unlocked and added to the player’s Arcade Garage. If the player is not successful, the car remains locked and players cannot use it in other parts of Arcade Mode. One challenging aspect for players is that the car, the track ribbon, and the target lap time to beat are all preset. Thus, if players would typically shy away from racing with the powerhouse Nissan Silvia Top Secret (a car not available until well into Career Mode) because of its notoriously intractable handling, in order to beat time trial #3, they would have to use that Silvia. In addition, it wouldn’t be the Silvia on an oval track, it would be on a challenging real-world race track (i.e., Tsukuba) and the lap time would need to be under 46 seconds to boot!

While the design intent behind the time trials was that they be challenging for players, the designers did not want them to be overly frustrating. Indeed, what could be more frustrating to a player than the inability to unlock a car for their arcade garage because of a target lap time they perceive to be impossible beat after the 50th unsuccessful lap? Luckily, the Forza 2 designers had a clear design intent for the time trials that we could use to test against: The time trials should be challenging to players but approximately 80 percent of the target users should be able to complete any particular time trial and unlock the car after ten laps.

15.3.1 The Problem

At the point in production that the twenty-five time trials were finally getting locked down, we still did not know whether the design vision for them was being realized.
The target lap times for each trial were determined by several members of the team but not tested with users. Although each trial target time felt challenging, but doable, by the members of Design and Test that helped determine them, these team members were no longer “typical players” for Forza 2. In fact, they had been racing on some of the same tracks with the same cars for several months during the game’s development. This fact presented us with a problem: we didn’t know how close the target times determined by the team members mapped onto the skill level of the typical player. If the target times were set too low, less than 80 percent of the target population would be able to complete each one after ten laps. Our research question became, “What percentage of typical Forza 2 players can complete each time trial after ten laps on a track?”

15.3.2 The TRUE solution

We brought people in and had them play through all twenty-five time trials. For each time trial, players raced ten laps, and we automatically recorded lap time, lap number (e.g., three of ten), and other contextual variables. We then calculated the percentage of players that beat the target lap time for each time trial. The results can be seen in Table 15.1.

As can be seen in the table, the design intent that we tested against was far from being realized in the target audience. The target times preset by the team members

<table>
<thead>
<tr>
<th>Trial #</th>
<th>Target Lap Time (secs)</th>
<th>Participants that beat after 10 laps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.81</td>
<td>24 percent</td>
</tr>
<tr>
<td>2</td>
<td>44.69</td>
<td>41 percent</td>
</tr>
<tr>
<td>3</td>
<td>45.697</td>
<td>0 percent</td>
</tr>
<tr>
<td>4</td>
<td>51.292</td>
<td>97 percent</td>
</tr>
<tr>
<td>5</td>
<td>55.602</td>
<td>7 percent</td>
</tr>
<tr>
<td>6</td>
<td>56.033</td>
<td>54 percent</td>
</tr>
<tr>
<td>7</td>
<td>57.132</td>
<td>37 percent</td>
</tr>
<tr>
<td>8</td>
<td>57.782</td>
<td>19 percent</td>
</tr>
<tr>
<td>9</td>
<td>58.731</td>
<td>59 percent</td>
</tr>
<tr>
<td>10</td>
<td>59.964</td>
<td>8 percent</td>
</tr>
<tr>
<td>11</td>
<td>62.8</td>
<td>8 percent</td>
</tr>
<tr>
<td>12</td>
<td>68.5</td>
<td>12 percent</td>
</tr>
<tr>
<td>13</td>
<td>73.8</td>
<td>19 percent</td>
</tr>
<tr>
<td>14</td>
<td>77.5</td>
<td>12 percent</td>
</tr>
<tr>
<td>15</td>
<td>78.5</td>
<td>12 percent</td>
</tr>
<tr>
<td>Trial #</td>
<td>Target Lap Time (secs)</td>
<td>Participants that beat after 10 laps</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>16</td>
<td>78.8</td>
<td>4 percent</td>
</tr>
<tr>
<td>17</td>
<td>83.4</td>
<td>0 percent</td>
</tr>
<tr>
<td>18</td>
<td>85.8</td>
<td>8 percent</td>
</tr>
<tr>
<td>19</td>
<td>98.3</td>
<td>0 percent</td>
</tr>
<tr>
<td>20</td>
<td>98.8</td>
<td>6 percent</td>
</tr>
<tr>
<td>21</td>
<td>101.8</td>
<td>29 percent</td>
</tr>
<tr>
<td>22</td>
<td>111.8</td>
<td>0 percent</td>
</tr>
<tr>
<td>23</td>
<td>118.4</td>
<td>30 percent</td>
</tr>
<tr>
<td>24</td>
<td>236.7</td>
<td>0 percent</td>
</tr>
<tr>
<td>25</td>
<td>409.6</td>
<td>0 percent</td>
</tr>
</tbody>
</table>

were consistently too difficult. In fact, only one time trial (that is, #4) met the design intent criteria. Indeed, six of the twenty-five time trials were unbeatable by players after ten laps.

### 15.3.3 Design Impact

Prior to our TRUE test, the target times for each trial were considered shippable. Based on the data from the TRUE test, we ended up changing the car used on some trials and the target time was adjusted on almost every single trial. Once again, the data collected from our TRUE test helped Design determine where the new target times should be set. Not only had we collected pass/fail info for each player with respect to each time trial lap, but we also collected their exact lap time. Utilizing these data we calculated the minimum, maximum, and average lap times for "passed" laps as well as "failed" laps which helped to determine where to place the new adjusted target time.

Had we not had the robust TRUE instrumentation in place and run the test, the typical player experience of the time trials in the released version of the game would be considerably different from what the designers intended. While the target times might have felt right for team members, they were too difficult for the typical users. Indeed, based on our data, there would have been time trials in which significantly large portions of the target audience most likely could not have passed. Yet, the testing to uncover these issues did not take weeks of planning, orchestrating, and analyzing of the data. Instead, because of where it happened in production, the uncovering of these issues and implementation of changes to fix them happened in a much abbreviated timeframe—a process that could have only been possible with TRUE instrumentation.
15.4 Shadowrun—Beta

There are times when the research questions we have require a large number of players—more than the fifty-one that we can simultaneously accommodate in our playtesting facilities. For example, we may need to test matchmaking systems that support thousands of simultaneous players, balance game economies, or tweak the attributes of character classes. In some cases, we can get the data we need from players individually interacting with a game. Sometimes, we need many—often thousands—of players simultaneously interacting with both the game and one another. In these situations, we beta test the game and collect instrumentation data from many thousands of players over an extended time. In this section, we discuss the use of instrumentation in the Shadowrun beta.

15.4.1 The Problem: Class Selection

Shadowrun is a multiplayer, round-based, first-person shooter game for the Xbox 360 and PC. Shadowrun allows players to choose their character class, skills, and weapons. At the beginning of the game, players choose from one of four different character classes, each with different abilities. Players can pick trolls who are strong but slow, elves who are quick and adept at magic, or dwarves or humans, each with their own special abilities. While this provided a lot of options to the player, it also created a challenge for the designers. Specifically, they needed to balance the game so that no character customization path would dominate while making all customization paths enjoyable to play.

We had many research questions for the Shadowrun beta, one of which we discuss here. We were interested in the popularity and effectiveness of different character classes. The number of combinations and permutations of classes in a game was large and potentially overwhelming to the test team, and so we used beta to test the popularity and effectiveness of character races with the players.

15.4.2 The TRUE Solution: Class Selection

We collected data from ten thousand participants who downloaded and played Shadowrun multiplayer from their homes with each other on their Xbox 360 using the Xbox Live online service. Using instrumentation we had a real-time stream of data about players’ choice of character classes and weapons as well as many other game events and behaviors. The game designers used these data to tweak game parameters and then deliver an updated version of the game to the participants. This iterative process of collecting and interpreting data and updating the game took place over the course of four months.

Our first research question concerned the popularity of the character classes. There were four classes in the game, each with different abilities. The designers’ intention was that each character class should provide strengths and weaknesses that
Character classes participants selected over the course of a month of the Shadowrun beta test.

differentiate it from the other classes and therefore appeal to different play-styles. To gather the relevant data, the game logged each time a player joined a game and chose a character. We were able to look at both the choices of each player in each game as well as the choices of each gamer across the many games they played. This way, we could determine both overall trends in choices as well as individual preferences for character races. We also logged the success of each player both in terms of how well they individually did in a game and whether their team won or lost.

We found, by tracking character choices, that over time one character class was clearly preferred to all of the others. If the classes were all as fun or effective, we would expect to see each being selected approximately 25 percent of the time. Figure 15.12 shows the actual character classes participants selected over the course of a month of the beta test. The Elf class, the top line in the graph, was significantly more popular than the other classes. The fluctuating popularity of the character classes shown on the left of Figure 15.12 (the beginning of the month) occurred during a period when we introduced 1000 new participants into the beta. The popularity of the races fluctuated as the new participants explored each of the races. A few days later, however, the preference for Elf once again emerged. This illustrates the advantages of beta testing over an extended period of time. A research methodology that limited our investigation to a few games—or even a few days of gaming—would have given us a misleading picture of character class preferences.

15.4.3 Design Impact: Class Selection

Over the course of the beta, the designers used these data to tweak the attributes of the character classes. Through several iterations of the game, the design team was able to realize their original design intention for a balanced selection of character
classes. Each character class had its strengths and weaknesses, but no class was clearly dominating the others.

Beta testing with instrumentation enabled us to tweak the class variables in the game to achieve the designers’ intent of having a game with well-balanced and differentiated character classes. Conducting the beta test over several months enabled us to collect data that showed both how new users approached the game as well as patterns in players’ behavior over the course of many game sessions. The large number of participants in the beta test also enabled us to examine the variables of interest with many different types of players in many different game situations. Further, being able to iterate on the game and continue testing with the same participants enabled us to verify design changes before the game was available to consumers, none of which would have been possible using in-house testing or without instrumented data collection.

15.5 Crackdown: Demo

In some cases, information gleaned from instrumentation of the game can be used to inform the creation of a demo. When done correctly, game demos can increase sales of the retail game. Moreover, the prominence of demos in console gaming is skyrocketing thanks to online distribution via Xbox Live or PlayStation Network. Yet, demos suffer from a lot of constraints—they are produced extremely late in the product cycle (sometimes after the retail code is finalized) by an understaffed and exhausted team, working on a tight deadline if the demo is to come out before the release of the game. Building a quality demo can be very challenging under these circumstances. As a result, leveraging information gained from instrumentation can be a boon.

15.5.1 The problem

Crackdown is an action game developed by Realtime Worlds and released in the spring of 2007. You play the role of a super-powered Agent who is capable of jumping tall buildings, throwing cars three city blocks, creating massive chain reaction explosions, driving powerful vehicles through crime-laden streets, all while battling gangs and taking out bosses in a giant urban sandbox. You start the game as a normally powered agent, but eventually level up your strength, agility, weapon proficiency, driving ability, and explosives destruction by collecting orbs associated with each trait. The game is sandbox-style, where users can go anywhere in the world and do anything they want right from the beginning.

The issue was how to capture the essence of Crackdown without giving away too much of the game. Sandbox games are not very amenable to parceling into demosized chunks. Moreover, the unique joys of Crackdown come from leveling up and experiencing the super powers the Agent gains—things that take hours of gameplay to achieve in the retail game. We were tasked with coming up with a demo that showcased the game and the Agent’s abilities that gave people a sense of what the game was like but leaving them hungry for more.
15.5.2 The TRUE solution

We did extensive instrumentation work on the retail version of the game to understand how people progressed through it, how many orbs they collected, when they leveled up, and so on. Further, we knew from the included surveys that the game went from being somewhat fun to being incredibly fun once people started leveling up their stats by collecting the orbs. Once people leveled up, they stopped viewing the game as another action/racing hybrid clone but the unique sandbox that is *Crackdown*—and they loved it.

15.5.3 Design Impact

The demo consisted of one (of five total) island in the game, so it included almost 20 percent of the content in the game. There was much concern that we were giving too much of the game away, so we knew we had to time limit the demo. But we didn’t want the clock to start ticking too soon, before people experienced the fun of the game. Our solution was to start a 30-minute timer, which started immediately after people leveled up for the first time. And to ensure that this happened quickly and that people got to the fun of the game right away, we had a dramatically accelerated skill progression curve. In order to create this accelerated skill progression curve, we looked at our instrumentation data and saw how many orbs of each type people had collected at the 10-, 20-, 30-, and 40-minute mark. We used those values to set the requirements for leveling up in the demo, then tested that demo to see what people thought. They leveled up right as we thought they would, and they absolutely loved the demo. They got a great taste of what it would be like to be a powered up Agent, but were hungry for more when the time ran out. This is exactly what we were hoping to achieve, so we shipped the demo with those settings. The *Crackdown* demo then went on to be the most downloaded demo in the history of Xbox Live at the time being downloaded more than 1 million times in the first month it was out.

15.6 Next Steps and Resources

15.6.1 TRUE Instrumentation and the Product Development Cycle

Instrumentation activities, like other major activities in the development process, take place at different points. Much of the work will be front loaded in the schedule. You should plan for this fact, but the good news is that the early work ensures that there’s less work later, when other demands on your time get really nasty. Below is a general list of the major “milestones” that should be reached at each point in the cycle. As always, these are guidelines rather than strict rules.
1. Preproduction
   - A high-level instrumentation plan is in place. This consists of the types of research questions, rough timelines, and agreements on roles and responsibilities.
   - The person who will be doing the instrumentation should start familiarizing themselves with the game code.

2. Production
   - A detailed instrumentation plan is in place a year prior to release. This plan consists of the research questions, the mock reports, a list of roles and responsibilities, and a timeline.
   - All of the main hooks are set by Code Complete. There may be some minor iteration of hooks post-Code Complete, but the bulk of the heavy lifting should be done.
   - Data collection starts, reports are iterated as needed. For some additional examples of reports generated from our TRUE instrumentation for Bioware, see DeRosa (2007)

3. Polish/Bug Fixing
   - Data collection continues at a frenzied pace, the game itself is iterated and verified to improve.
   - If you’re doing an instrumented beta, it probably happens here.
   - Any work on the demo also occurs here.

4. Post-release
   - In the case of key franchises, you may want to collect some data to better understand these franchises.
   - The data will be a small set, and no attitudinal data should be included
   - It is possible to use these data to refine a game you have already released, as in http://www.steampowered.com/status/ep2/ep2_stats.php
   - It is desirable to use these data to benefit your audience in the form of stats pages, as in http://www.bungie.net/

15.7 Lessons Learned
Successful instrumentation studies are complex and difficult to execute well. The instrumentation infrastructure for a game is built on a changing platform—that is, the game itself. The number of variables collected, the consistency with which states are defined, the knowledge of game and the programming expertise of the
instrument builders all play a role. Below we outline some of the lessons we have learned over the years of conducting successful instrumentation projects.

15.7.1 Lesson 1: Plan for instrumentation early, make sure there is time for iteration

Instrumentation can be a time-consuming (but rewarding) effort, involving a variety of resources at different points in the game development lifecycle. As such, it needs to be planned and scheduled, like any other project in the game. Stakeholders need to be identified, timelines need to be spelled out, and dependencies need to be identified early. If the instrumentation effort is not well planned and is pulled together right at the end of development, it is likely to cause more problems than it solves.

One of the biggest things to be sure to schedule for is ample time for iteration. There needs to be iteration of the research questions, iteration of the reports, iteration of the variables that are tracked, and most importantly, iteration of the testing.

The point of doing instrumentation, indeed any form of user research, is to improve your game. It is not enough to simply understand what problems people are having—you need to address those problems and then verify that those fixes work. The Halo 2 example cited earlier illustrated that we not only identified a problem, but that we made significant gameplay changes, tested again, and saw that those problems went away without introducing any new problems.

A testing schedule where you test over the weekend, make changes during the week, and retest over the following weekend is common during the final balance and polishing stages of game development.

15.7.2 Lesson 2: Start with your research questions

Although this may be obvious, many first-timers overlook this step. There is a temptation to automatically start thinking about what variables to track because those variables are typically more concrete and easy to think about. It is more difficult to contemplate what questions you are trying to answer, and then go to the information that is needed to answer those questions. However, clearly identifying the research questions ahead of time will ensure that you are collecting the data you need—and not wasting time collecting information you will never use.

Identifying the research questions should be a multidisciplinary effort. The Design team is a good source to tap, as they are likely to have lots of questions and will be the ultimate consumers of the information produced. The Test/Quality Assurance team is another good resource as they are intimately familiar with the various nooks and crannies of the game as well as sensitive to balance issues due to their constant playing of the game. Producers and marketers can frequently identify broad questions based on the vision of the game and the needs of the target audience. And, naturally, if a User Researcher is available, they should be involved as their whole discipline involves turning vague research questions about consumer interactions with the product into actionable data.
15.7.3 Lesson 3: Keep the number of variables you are tracking to an absolute minimum

Once you start thinking about your research questions, it becomes tempting to track everything. The thought process that leads to this temptation makes perfect sense—in order to understand a system, it is natural to want to track all of the inputs into that system. For example, if you wanted to understand what weapons people used in a role-playing game, you might think that weapon choice is a function of weapon availability, which is a function of the amount of money and the cost of the item in the local store, which is a function of how the NPC shopkeeper feels toward the player, which is a function of the player’s stats, which is a function of what quests have been completed, which is a function of ... well, you get the point. However, collecting data on too many variables causes all sorts of unintended consequences, such as:

- More time spent hooking the variables
- Decreased reliability of hooks—setting hooks is as much as an art as a science, and errors are common. The more hooks, the more chance of error. And errors in one variable cause people to question the validity of all of the data.
- More time spent testing hooks to make sure they are recording data correctly
- More reports to create, review, and understand. There is increased likelihood of losing sight of real problems in the sea of data

Rather than tracking everything, you should limit yourself to the handful of variables (and related contextual information) needed to know whether there is a problem. For example, if you want to identify problematic combat encounters, all you need to track are deaths, cause of death, and how the player felt about the death. That is sufficient to say “there’s a problem here,” and you could review the synced video in order to understand why that problem is occurring. Depending on the game, we typically recommend that you track no more than 15 different events, along with related contextual information (timestamp, x and y coordinates, etc.). When determining what variables to track, the maxim “less is more” truly holds.

15.7.4 Lesson 4: Build sample reports BEFORE you set your hooks

Hooking your variables of interest is time, and developer, intensive work, so you should take steps to prevent unnecessary rework. Even if you start with your research questions (Lesson 2), it is easy to forget to include certain hooks or to request tracking variables that you will never actually use. One of the best ways to avoid these pitfalls is to mock up reports populated with fake data, and then have all interested parties (Design, Test, Producers) review the reports to see if they contain everything that is needed to identify problems and make needed design changes. Inevitably, once reports are mocked up and passed around, people identify
missing variables, come up with new research questions, or concede that certain pieces of information they thought would be important are actually not needed.

Other benefits of mocking up reports before setting the hooks include:

- Helping the developer who is setting the hooks make decisions on how to set the hooks. Frequently, developers can set hooks in multiple ways, including ways that affect what the data mean. When they know what you are trying to achieve, they can make better decisions on how to set the hooks.
- Getting buyoff from the stakeholders on what the instrumentation effort will yield. For many people, seeing is believing.
- Quicker turnaround of reports after data collection. If you mock up your reports in your reporting tool, you can simply change the data source from the fake data to the real data, and your reports are instantly updated. If you wait until the actual time of data collection to build your reports, there will be a substantial delay between when you collect the data and when you can actually use it. At this phase of the game lifecycle, time is the most precious element, so quick turnaround of data is essential.

15.7.5 Lesson 5: Represent the data visually

There is so much data to consume, you need to build reports that are easy to understand. One of the most powerful visualizations is a map of the level, upon which your data can be displayed (see Figure 15.6). For example, this image displays the unit deaths of a person playing a real-time strategy game.

In order to display data on top of a map, you need to:

- Get a screenshot of the level
- Note the x, y coordinates of all 4 corners
- Record the x, y coordinates of every event (as part of your contextual data)
- Use your data visualization tool to plot the event x, y coordinates relative to the coordinates of the image

15.7.6 Lesson 6: If possible, have Design specify their design intent so we can compare actual with intended performance

Even if you include in-game surveys, it can be difficult to identify whether there are subtle problems with progression or level design. It can be very helpful to have the Design team specify what sort of experience they were trying to create at different parts of the game. Depending on the designer, this could be very general ("ok, so they just defeated the Mega Boss, so this part of the game is supposed to be a bit of a rest. People should be exploring here, but not dying too often") or very specific ("I
want 90 percent of people to get a lap time of 2:00 or less within 5 attempts in a D class car or 3 attempts in a C class car." (Romero, 2008). Knowing the design intent vastly simplifies issue identification. It is possible to build reports that show deviations from expected levels of performance, quickly highlighting problematic areas.

Even if design intent is not explicitly stated, designers implicitly know how they want their game to be played. Designers are so intimately familiar with the game and the experiences they are trying to create, they can often spot problems in the data that others overlook. For this reason, designers should always take the time to review the data after each test. And because they will be doing so, it is vital that Design be involved in focusing the research question (Lesson 1) and reviewing the reports (Lesson 4).

### 15.7.7 Lesson 7: Test to make sure your instrumentation is recording data reliably

Game code is constantly changing during the production phase of development when you are likely to be doing instrumentation. Thus, hooks that were working in one build can be broken the next—either not recording data, or recording nonsensical values. Nothing damages the overall credibility of instrumentation more than producing data that are obviously wrong—faith in the entire system is extinguished if even one piece of data is wrong.

As a result, testing the build is vital. You simply do not want to discover an error in your data collection after twenty people play your game for twenty hours over the weekend. Having the Test/Quality Assurance department verifying the instrumentation is working correctly is vital before any data collection.

### 15.7.8 Lesson 8: Integrate into the source tree so you always have an instrumented build

Integrating the hooks can either be done by hand every time you need it, or done once and checked into the source tree so that it is included with every build. We strongly recommend integration in the source tree so that you always get a build (saving time and reducing the risk of having the build you include hooks in not working properly). Time is tight, so any steps you can take to increase the turnaround and flexibility of your instrumentation is worthwhile.

### 15.7.9 Lesson 9: Instrumentation does not replace other forms of getting feedback

Instrumentation is very well-suited for collecting data about how people perform in a game over extended periods of time, but it doesn’t replace the need for detailed research on the initial gameplay and core mechanics (typically done in usability) or
collecting attitudinal feedback on what people think about different aspects of the
game (such as in playtests).

In reality, instrumentation can work hand in hand with other user research tech-
niques (Kim et al., 2008). For example, we used instrumentation and usability test-
ing together to improve a real-time strategy game. In this series of studies, people
played the single-player campaign over the course of a weekend. The single-player
campaign consisted of a series of missions which had to be completed in order for
the game to progress. We used instrumentation to get a rough understanding of
which missions were most problematic, and then did a detailed follow up usability
study on only those problematic missions. We simply did not have the time needed
to do the detailed usability testing of every single mission. By doing the instrumen-
tation, we were able to identify which missions were most problematic and focus
our research energies there.

15.8 References

game: The consumer playtest in game design. *Game Studies: The International Journal of
org/0501/davis_steury_pagulayan/
tracking_player_feedback_to_.php
OR: Intellect Books.
Fulton, B. (2002). Beyond Psychological Theory: Getting Data that Improve Games.
*Proceedings of the Game Developer’s Conference, San Jose, CA.*
www.steampowered.com/status/ep2/ep2_stats.php
research. *Workshop at the AAAI (American Association for Artificial Intelligence)
Real-Time User Experience (TRUE): A comprehensive instrumentation solution for com-
plex systems. *Proceedings of the SIGCHI conference on Human factors in computing sys-
tems, Florence, Italy.*
Evaluation Method: Better Products in Less Time. In R. Bias, & D. Maybews (Eds),
*Cost Justifying Usability: An Update for the Internet Age* (pp. 489–517), NY: Morgan
Kaufmann.

264


