Visualizing Game States

Steven Brener

Abstract— **The scale and complexity of games have increased to a point where the older methods of testing are no longer sufficient. A new method of testing using large amounts of data collected by players has become a common way to supplement the older methods of game testing. The most common way to use this data is in the form of visualizations. This paper presents a visualization of game data centered around user interactions with the game that can be used by game developers during the testing process. The visualization allows users to view the events that took place over the course of multiple games, explore more specific parts of the data using a variety of filters, and replay a series of game states that took place in a single game. The data used in this paper was collected from personally playing several rounds of Quake 3, which is a multiplayer first person shooter [6]. The amount of data is limited, and every round was played against an A.I. controlled bot. To provide examples of situations this visualization can be used in, a few of the games were played normally, and a few were played with special conditions to force certain patterns to appear. The current visualization is still a prototype. More data needs to be collected and more evaluation needs to be done to determine how effective it will be in practice.**

I. Introduction

Testing is an important part of a video game's development cycle. Game developers spend a lot of time and money on the testing process, which has resulted in a rise in the amount of research in the area. When a game is released, the expectation of customers is that it will run smoothly with few or no errors, and that playing it will be fun. Like all software, games can have a wide variety of bugs that will have a negative impact on a user's experience. The complexity and high level of interaction in video games make finding all of the problems especially difficult. Video games can also have usability problems that aren't necessarily bugs, but still make the game less enjoyable for players. Testing is the main way that game developers find and eliminate these problems, so that the final game will meet the expectations of the players. Finding problems related to user experience requires the ability to see how users interact with the game. As a result, testing is usually done by getting hired testers or volunteers to play the game [10, 7].

Recently, game developers have also begun to use another method of testing, called instrumentation. While people are playing the game, data related to the players interaction with the game is collected. Collecting the data automatically while the game is being played is faster and more accurate then the normal method of testers reporting what they find [8]. This data can then be analyzed to see how the players are interacting with the game and find ways that the game can be improved. Almost anything in a game can be measured, but some data is more useful than others. Some of the most common types of data that are collected in practice are the time spent to do something, position at each point in time [1, 4], deaths [3, 8], and other, possibly game specific, events such as item pickups or uses [9]. One of the simplest ways to analyze this kind of data is to focus on a single type of data, like player deaths, and represent it as a bar chart. If there are an unusually high number of deaths in one area, then it might mean it's too difficult and should be made easier [8]. Without any context, this method of analysis can find where a problem might exist, but isn't able to explain what's causing the problem.

The increased use of instrumentation in the testing process for games has led to new methods of analyzing this data. These methods usually involve new ways of visualizing the data collected from games. The large quantity and variety of data involved makes visualizations of that data necessary. The most common visualizations are bar charts and heat maps [3, 8] which can also be used as supplemental views in more complicated systems. More complicated and specialized visualizations can not only identify potential problems, but also help developer's find and fix the cause of the problem by providing additional context. These visualizations usually involve displaying various events in the game on an overlay of the map [9, 5]. By combining more than one type of data in a single view, it's possible to get a better understanding of how the players are interacting with the game. This makes it easier to find the cause of problems, and identify ways in which players behave differently than intended.

In this paper I present a visualization designed to be used by developers during the testing process. Like many of the more specialized visualizations, mine focuses on the time and position of player related events. The positions of events over multiple games can be used to find general patterns relating to the movement, deaths, and items pickups of players on a map. A time filter can be used to replay a sequence of events that occurred in the game. By doing this, a user can get a better understanding of what interactions in the game led up to an event. The goal is for developers to be able to combine different views and filters in this visualization to find problems, or unexpected behavior and then use this information to improve the game.

II. Related Work

A wide variety of visualizations have been presented in the past to represent different types of game data. The most

common and most generic of the visualizations usually revolve around bar charts and heat maps. Other, more specialized visualizations, provide additional context that can be used for a more in depth analysis of how a game is being played.

The bar chart and heat maps are the easiest visualizations to make, but provide the least amount of information. One example of heat maps and bar charts being used in practice is from Valve. Data was collected from people playing Half-Life 2 episode two and used to make several visualizations [2]. Completion times and number of deaths are presented in bar charts, and the locations of deaths on each map are shown using heat maps. Using these visualizations can answer simple questions such as, where are people dying the most, and how long does it take to beat the game. However, without additional context, it is impossible to determine why people are dying in certain places, or why it takes a certain amount of time to complete each area.

Many of the visualizations try to add the cause of death to provide some additional context. One system developed by Microsoft Game Labs does this by starting with a bar chart showing the number of deaths in each mission of Halo 2. One mission had an unusually high number of deaths. The data was then narrowed down to the percentage of deaths from each cause in a specific part of that mission. This was improved even further by having a video clip for each death that allowed them to watch how the deaths were happening. Doing this required looking through multiple levels of bar charts, which takes time. The videos used at the end were still needed to explain the data in the end, and recording videos of every event in a game isn't ideal [8].

Another method of including cause of death is to add them to heat maps that also show positional data. Two papers by Drachen and Canossa [4, 3] do this in different ways and also provide a path based visualization. In one of the papers, the cause and position of death is recorded for a shooter called Fragile Alliance and multiple visualizations are made showing the deaths from one cause each. The second paper shows a variety of heat maps based on data from Tomb Raider: Underworld that focus on different aspects of a death. One uses a scale from green to red to show the number of deaths in an area, while another overlays multiple heat maps on top of each other for different causes of death. These make it possible to determine how people are dying, but overlaying multiple heat maps can make it more difficult to understand what's happening in each area. The first paper also provided a visualization of the paths people took while playing Kane & Lynch. The position of players at each point in time was recorded along with some other information. The points alone were drawn on the map along with a path that the map designer considered to be the perfect path through the level.

Many other visualizations also include data related to the path a player takes through a map. One tool developed by Chittaro et al.[1] called VU-FLOW, focuses on displaying paths. The tool can display paths of individuals and groups in multiple ways that are useful for generic analysis of movement on a 2D map. Exact paths can be shown for individuals and aggregate visualizations are available to visualize the paths of groups. Although no specific game data or events are shown, the techniques in VU-Flow are generic enough that they could be applied to most navigation data, including data collected from video games.

The most specialized visualizations attempt to combine everything in a way that allows developers to replay the actions taken by players. One visualization presented by Hoobler et al.[5] called Lithium, focuses on a high level view of player interactions and team strategies in Return to Castle Wolfenstein: Enemy Territory. The visualization includes a large variety of data including both individual interactions and team overviews. Individual data includes positions of players, paths taken, deaths, and other useful information for following a specific player through the game. Team data like occupancy, support fire, and medic efficacy can also be displayed to determine how well a team plays. This visualization targets spectators more than developers, but several of the techniques used could still be useful for analysis during development.

Another specialized visualization presented by Moura et al.[9], focuses on displaying player actions in Dragon Age Origins. The amount of time spent in an area, and the amount of time spent performing different types of action in an area are displayed using a circle of increasing size for each part of the map. Aggregated lines show the paths taken between areas to provide an overview of a player's movement. It's also possible to see which items were picked up and which NPCs were talked to in each area. This visualization is helpful for seeing how players interact with different parts of the map, but doesn't include combat data, and doesn't display the exact location of events and paths taken.

III. Description

The visualization I've made is centered around the positions and times of player related events. An overview shows all of the events at once using a map overlay of the events, a time line of events, and bar charts to show total counts of types of events. This overview can be used to see general patterns across multiple games. Filters on the time, types of data, and specific games can be used to focus on specific parts of the data. At the most detailed level, it's possible to reproduce the sequences of events that occurred in an individual game.

The data used to test this visualization was taken from Quake 3. The game was modified so that the details of

certain events would be recorded as the game was played. Whenever a player dies, picks up an item, or spawns, the time and location of the event are recorded, along with additional data relevant to the event. The position of every player was also recorded every second of the game. As of now, the data collected is from only four rounds of the game. Each round was played by myself against an A.I. controlled bot, on the first non-tutorial map of the game. Two of the rounds were played normally. One round was played with the strategy of camping in a specific location. The final round was played after modifying the game to simulate the effects of cheating. This round was not included in most of the analysis because of how biased the resulting data was. The data was stored using csv files and the visualization was implemented using D3.

A. *Overview*

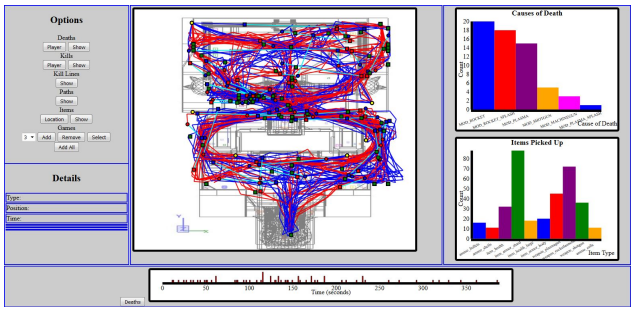

Fig. 1. Initial View

When the visualization is initialized, it presents all of the data given to it using a map overlay, a time line, and two bar charts. Figure 1 shows the initial view of the visualization using the data from the first 3 games. The map overlay is the central focus of the visualization. The exact positions of all events are recorded when collecting data. This is used to display the position of everything that happened on the map. The deaths are represented as circles that are colored based on either the player that died, or the cause of death. The kill positions are represented as squares with the same coloring scheme. A line connects a death with the associated kill, and all three are considered to be the same event. Items are represented as green squares. To prevent confusion, green is reserved for items and cannot be used as the color for kills. Spawn points are represented as yellow circles, and yellow is reserved similarly to prevent confusion with death circles. The paths taken by players are drawn with lines connecting the consecutive positions of each player. The lines are colored using the color assigned to the player. This initial view gets cluttered quickly but gives a good sense of the maps structure and where most of the events are happening.

A time line at the bottom of the screen shows how many events of a certain type occur at each second of the game. A bar for each second shows the count of a specific event type that occurred during that time. The initial view shows how many deaths occurred a different times throughout the game, because these are the most meaningful events. It can also be switched to a view that shows when items are collected. The final event shown on the time line will always be the last death in the longest game. This makes it easy to see the maximum length for a game in the data. The time line view can be used to find the times when most of the events are occurring.

The final part of the overview is made up of bar charts for each type of event in the game. The only events shown in this visualization are deaths and item pickups, but bar charts could be made for any type of event. Each bar chart includes a bar for each type of event. For deaths, the categories are based around the cause of death. For items, they are based around the type of item. The bars in the death bar chart are colored based on the color assigned to each cause of death. The events displayed on the map can be colored using the same scheme. This makes it easy to connect bars to the the specific locations of the events associated with them. Since items are consistently colored green, the bar colors have no additional meaning in the item chart. Both bar charts make it easy to identify weapons or items that are being preferred by the players.

B. *Filters*

Filters are used to narrow down searches and make the main view more readable. The first type of filter targets the types of data shown on the map overlay. Each type of symbol can be faded out to make other data more readable. The death, kills, lines connecting deaths and kill, items, and paths can all be removed from the map separately and then brought back to full view by clicking buttons on the side. The way deaths, kills, and items are colored can also be changed. The deaths and kills can either be colored based on the player that was at that location during the event, or based on the cause of death associated with the event. The items can either be colored green to show all locations equally, or colored from white to green to show which items were picked up the most often. These options can be used to focus on the data that the user is interested in at the moment. For example, if a user wants to see where all the deaths are occurring, they don't need to see all of the paths.

The bar charts can be used to filter data based on the value of the data. If the bar associated with deaths caused by rockets is clicked, then rocket data will be highlighted and everything else is faded out. The death events where the cause of death was a rocket, which include the death location, kill location, and line between them are highlighted yellow to make them easy to see on the now gray background. The time line is also changed to only show the times of deaths that were caused by rockets. These filters can be used to focus on specific causes of death or

items that stand out in some way.

Everything can also be filtered based on the game the data is from, and the time it occurred. A selector in the options menu lets users remove games from the view, and add them back in. Events from different games don't affect each other, so this is useful when a user wants to see what happened in a game. The time line can be used to filter events based on the time they occurred. This can be used with the main view to see patterns that occur at times where a lot of events are occurring. It's most useful when combined with the game filter. By choosing a specific game and then filtering based on time, a user can reconstruct what happened at that point in the game. By moving the filter around, the user can reconstruct the entire game. Watching everything that led up to a death makes it easier to understand why the player died. Reconstructing the entire game makes it easier to see how the players involved interact with the game in general.

IV. Uses

The visualization I've made is intended to be used by game developers as part of the testing process. The game data presented in the visualization is collected while people are playing the game. The automatic collection of data means that the data can be collected from not just professional testers, but anyone that plays the game. This makes it easy to collect large amounts of data at any stage of testing. The data is based on player related events, and can be used to understand how players interact with the game. By combining information from the three views, and the filters used in the visualization, developers can find a wide variety of problems and ways to improve the game. One of the simplest ways to do this is to look for unusual patterns that appear across all games. Zooming in on a specific player's actions can provide additional context that helps with understanding why certain patterns exist. Additionally, it could be possible to use the visualization after development is complete to identify cheating. Although the only data that has been used in the visualization so far has come from Quake 3, I think the visualization can be used for most other games as well. Usable data can be collected from any game in which players move around a map and trigger events.

A. Finding General Patterns

By viewing data from a large number of games, it's possible to identify patterns of interactions with the game that many players follow. These patterns could involve the way players move around the map, the weapons they prefer to use, the items they pick up, or even combinations of interactions that might suggest certain strategies are being used. Some patterns can be seen quickly using the bar charts, while others might require additional exploration using the filters. One way these general patterns can be used is to find problems where something is happening a lot more or less than is expected. If a weapon is being used almost exclusively, or an item is never being picked up, it could be the result of an imbalance in the game. If certain parts of the map are never being visited, it could mean that they are hard to reach, or something about that part of the map discourages people from fighting on it. If these types of problems exist, they would be clearly visible as extreme differences in bar chart heights, and areas of the map with little or no movement data.

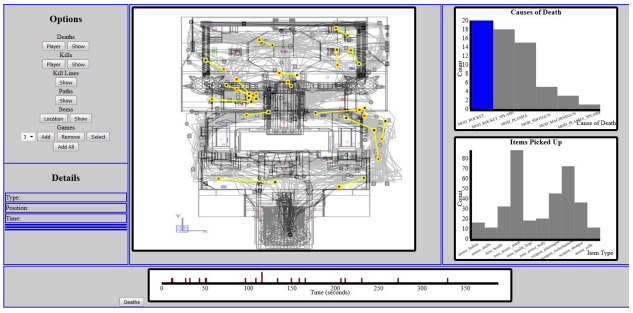

Fig. 2. The rocket launcher cause of death has been highlighted in this view.

An example of this can be seen in the data collected while I played Quake 3. By looking at the bar charts it becomes clear that the rocket launcher is the most used weapon. The cause of death from the rocket is divided into direct hit kills and splash damage kills, but both separately are still higher than the plasma gun kills and much higher than any other cause of death. Clicking the bar for the rocket launcher deaths highlights their locations on the map, as shown in Figure 2. Although there are rocket launcher deaths throughout the entire map, most seem to occur in the top half. This could be because the rocket launcher spawns on that part of the map. The reason for the high number of rocket launcher deaths in this cases is my personal preference for the rocket launcher. If this pattern existed in a larger group of games with more players, it could be an indication that the rocket launcher is overpowered compared to the rest of the weapons available on the map. If this wasn't the intent of the developers, then they would be able to see this imbalance quickly and fix it if necessary.

B. Reproducing Game States

One of the most useful features of my visualization is the ability to replay sequences of events. This is made possible by recording the exact times and locations of every event. Replaying the events is done by using a combination of the time and game filters. For a sequence of events to be meaningful, the events need to be from the same game. If multiple games are being shown, it becomes difficult to determine which events are actually connected. When two players approach the same corner from different directions, the expectation is that it will result in a shootout, and eventually a death. If the players were from different games,

then seeing this will be misleading, because they never actually came into contact with each other. This makes it necessary to focus on a single game before trying to reconstruct a specific sequence of events. Once a game has been chosen, the time filter can be set to any length of time greater than one second, and moved across the time line. Events will appear in order as they enter the currently selected time range and disappear again as the filter is moved past them. The resulting replay can be used to see the events leading up to an event of interest, the events that took place during a specific player's life from spawn to death, or the events of the entire game in order.

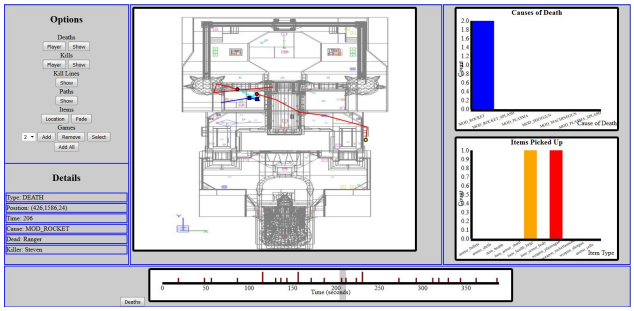

Fig. 3. This view shows a sequence of events that involve getting two kills by camping. Only data from one game is being shown, and the time filter has been set to extend from right before the first kill to right after the second kill.

Replaying events in a game allows users to answer question about why something happened, or why a pattern exists. For example, in one of the games I played, I spend a lot of time camping in the area near the shotgun spawn. This pattern is visible in the overviews, but can't be explained without reconstructing the events in the game. Just by looking at the initial view, a noticeable cluster of deaths can be seen in the area near the shotgun spawn at the center of the map. By selecting the game where I was camping, the pattern becomes even more obvious. The time filter can be used at this point to figure out why this cluster is appearing. Clicking one of the events in the cluster gives the user a time in the details table that can be used to find the event on the time line. By making a filter that extends a little further back from this time, it's possible to see what happened. Figure 3 shows the visualization after the game and time filters have been used to focus on the events surrounding one of the kills in the cluster. The enemy player spawned and headed towards the area of the map I was waiting at. During this time, there is a lot of movement from the red line which represents the enemy, but very little movement from the blue line used to show my movement. As the enemy approaches he is killed by a rocket launcher. A similar sequence of events, where I don't move much, and then get a kill in that area can be seen in several other deaths in the cluster. Even without already knowing that I was camping, I think someone would be able to figure it out by replaying the events the same way I have described. This strategy of replaying game states makes it possible to not only see that an unusual cluster of deaths appears on the map, but also come up with an explanation about why this cluster exists.

C. Detecting Cheaters

Even after a game has been shipped, this visualization could have some use as a way to detect cheating. Once the game is no longer being modified by the developers, there should be some consistency in the visualizations. For example, if over the course of hundreds of games with different players, a weapon is almost never being used, or has a consistently short or long range, there's not reason that this should change. It would be suspicious if a sniper rifle was suddenly the most used weapon in a game, and capable of consistently getting kills at all ranges. To detect cheating, a baseline is needed. A large amount of data from games that are known to be fair can be used for this. The resulting visualization can be compared to visualizations of games where cheating is suspected. If the general patterns in the two visualizations differ significantly, then there might be something fundamentally different about the two games. This can be analyzed further by looking at the events related to the differences to verify whether or not the differences were caused by modifying the game.

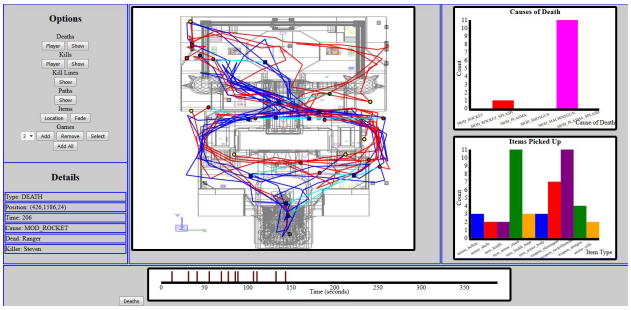

Fig. 4. This version of the visualization only uses data from the game played after modifying the amount of damage done by the machine gun.

To provide an example of this, I played a round of the game after modifying the amount of damage the machine gun does. After the modification, every bullet from the machine gun, which is fairly accurate and has a high rate of fire, was a one shot kill. The visualization created from this game, and shown in Figure 4, is significantly different from the visualization created from the combination of unmodified games in Figure 1. The bar chart shows that the main cause of death in the new visualization is the machine gun. There were very few machine gun kill in all of the other games combined, so this is a clear sign that something is different. This difference was the result of me only using the machine gun, which could have happened in an unmodified game as well. However, it would be much less likely that every single death was caused by a machine gun in this case. The enemy bot would have gotten more kills as I tried to stick to a much weaker weapon.

Another difference is visible on the time lines. The last death in the modified game was much earlier in the time line than in the longest game. This was because of how much faster I could get kills using the modified machine gun. Although differences in visualizations might not be conclusive evidence of cheating, it provides a good indicator that something might be wrong.

V. Future Work

This visualization is still a prototype. It still needs to be polished, additional features can be added, and it currently doesn't scale well. The way events are represented on the map can probably be improved. Using simple shapes like circles and squares was the simplest way to do it, but more complicated representations could help users distinguish between items and kills, and spawns and deaths. Using reserved colors to create this distinction isn't ideal. Another problem is caused by the way the map is drawn in the background. The map image isn't connected to the data in any way. It serves its purpose of adding positional context to the data, but nothing is aligned correctly. These, as well as many other problems, should be addressed in a final version of the visualization.

The visualization could also benefit from the addition of more features. The paths in particular are difficult to read, because of how much data is used for them. Some of the techniques used in the VU-Flow [1] could help with this. The positional data used in my visualization is similar to the data used by VU-Flow. I think many of the aggregation methods presented in the paper could be applied the data I've collected as well. Adding ways to aggregate path data would improve the users ability to understand how players as a group move through the map.

Heat maps could also be a useful addition in the form of an alternate map view. Heat maps result in a loss of context, but could be used in a way that is similar to how bar charts are used in my visualization. Areas with a lot of deaths would be easier to find using a heat map once more data has been collected. After finding an area of interest, the view could be changed back to the current one where exact positions are shown. At this point, the features described above could be used to explore the data in that part of the map more. Adding a heat map or other aggregation techniques to my visualization would improve the users ability to find interesting patterns in the data to analyze further using the additional context provided.

So far, my visualization has only been tested using data from at most 4 games. Even with that small number, it's clear that showing the detailed view of every event isn't going to scale well. The use of additional aggregation techniques like heat maps, and the paths presented in VU-Flow [1] will help with this significantly. If the higher level overviews are easier to use, then the detailed view that provides the most context might not need to scale. Showing every event individually at the location they occurred at might not scale well in any representation. This detailed view is more suited to analyzing individual games, or small groups of games.

Once the visualization has become more complete, it will also need to go through some kind of evaluation, with much more data. The examples described above were based on a small number of games, and I forced certain things to happen. This shows what the visualization might look at in those situations. It doesn't show what it would look like when a large amount of data from many players is used as intended. I'm also the only one that used the visualization for analysis. To test it, a user study with game developers using it to analyze their games would be needed.

VI. Conclusion

Although still a prototype, the visualization I've presented appears to be useful for analyzing game data. The three views can be used to get an overview of what events occurred, as well as where and when they occurred. The filters make it easy to narrow down searches and focus on specific parts of the data that might be involved with a problem or possible improvement. Most importantly, the time filter can be used to replay game states and get a high level of context. This is something that many other visualizations with the same purpose are unable to do. By replaying the game states, with all of the context provided by my visualization included, users can come up with more detailed explanations of patterns seen in the data. Once completed, I think my visualization would be of practical use to game developers during the testing process.

References

[1] Luca Chittaro, Roberto Ranon, and Lucio Ieronutti. Vu-flow: A visualization tool for analyzing navigation in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on*, 12(6):1475–1485, 2006.

[2] Valve Corporation. Half-life 2 episode 2 stats, Accessed: 9/12/15. `http://steampowered.com/status/ep2/ep2_stats.php`.

[3] Anders Drachen and Alessandro Canossa. Analyzing spatial user behavior in computer games using geographic information systems. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, pages 182–189. ACM, 2009.

[4] Anders Drachen and Alessandro Canossa. Towards gameplay analysis via gameplay metrics. In *Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, pages 202–209. ACM, 2009.

[5] Nate Hoobler, Greg Humphreys, and Maneesh Agrawala. Visualizing competitive behaviors in multi-user virtual environments. In *Proceedings of the conference on Visualization'04*, pages 163–170. IEEE Computer Society, 2004.

[6] id Software. Quake 3 arena. PC, 1999.

[7] Katherine Isbister and Noah Schaffer. *Game usability: Advancing the player experience*. CRC Press, 2008.

[8] Jun H Kim, Daniel V Gunn, Eric Schuh, Bruce Phillips, Randy J Pagulayan, and Dennis Wixon. Tracking real-time user experience (true): a comprehensive instrumentation solution for complex systems. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 443–452. ACM, 2008.

[9] Dinara Moura, Magy Seif el Nasr, and Christopher D Shaw. Visualizing and understanding players' behavior in video games: discovering patterns and supporting aggregation and comparison. In *ACM SIGGRAPH 2011 Game Papers*, page 2. ACM, 2011.

[10] Charles P Schultz, Robert Bryant, and Tim Langdell. *Game testing all in one*. Course Technology, 2005.

[11] Günter Wallner. Play-graph: A methodology and visualization approach for the analysis of gameplay data. In *FDG*, pages 253–260, 2013.