# – Game Flow Analysis

## Introduction

Methods for automatic logging of gameplay data have become an important component of game design in the last few years. The collected data can be used for different purposes, such as analyzing behavior, identifying strategies, detecting bugs, game testing, balancing the game experience, classifying users, and understanding common behaviors. However, the analysis of the collected data can be challenging due to its volume. Thus, data visualizations have become a promising solution for exploring and understanding games.

The visual data mining approach is useful when the designer is not familiar with the gathered data or has vague analysis goals. Therefore, visualization methods are gaining popularity among designers for understanding gameplay data in the game industry. For example, a common visualization method is heatmap, which uses colors in a two-dimensional map to reflect density of certain variables in particular locations of the game. Recently, BioWare used heatmaps to analyze common bug locations (ZOELLER, 2010), while Valve used heatmaps to analyze multiplayer maps in Team Fortress 2 (AMBINDER, 2009). Meanwhile, Bungie and Microsoft used heatmaps to determine common places where players died in Halo 3 (ROMERO, 2008; THOMPSON, 2007). Figure 1 illustrates an example of heatmap usage in the *Unreal Engine*.

Another common usage for game data logging and visualization in the game industry is aiding play testing and game refinement (FULLERTON; SWAIN, 2008). Recently, game designers began to use statistical techniques to gather player data. For example, DeRosa (2007) described how BioWare used statistics during playtesters to determine where players spent their time and which special powers were used. Other researchers also tried to analyze movements during battles (HOOBLER; HUMPHREYS; AGRAWALA, 2004) and identify player behaviors (DIXIT; YOUNGBLOOD, 2008).

Thus this chapter aims at describing some approaches related to data logging and visualizations. The criterion used for selecting the approaches is similar to snowballing sample. We started with a recent graph visualization approach to describe gameplay data (WALLNER, 2013). Then we checked possible candidates based on its references list and selected another graph based visualization (WALLNER; KRIGLSTEIN, 2012). Continuing this procedure, we selected an approach that combines behavioral and contextual data visualization (KIM *et al.*, 2008) and a proposed framework for gameplay data logging (JOSLIN; BROWN; DRENNAN, 2007).

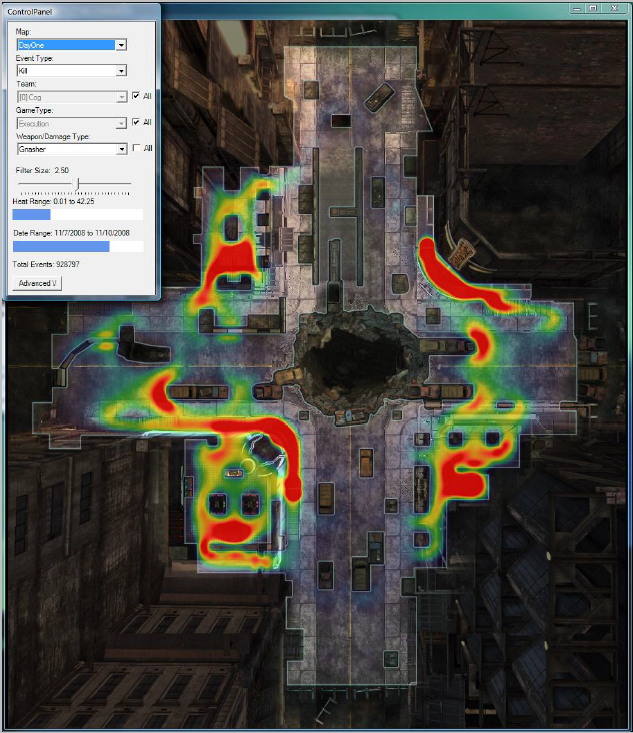


Figure 1: Heatmap representing shotgun kills on day one of Gears of Wars 2.

This chapter is organized as follows: Section 2.2 describes a gameplay data logging framework proposed by Microsoft. Section 2.3 describes a gameplay analysis of player behaviors using contextual data. Section 2.4 and 2.5 describes two visual tools designed to visualize gameplay data by using graphs. Lastly, Section 2.6 presents the final considerations of this chapter.

## Gameplay Data Logging

Joslin (2007) proposed a framework for gameplay data logging that uncovers gameplay events by attaching logging methods in game objects responsible for generating events during the game. The logging method gathers information according to an event model, describing which attributes and information are logged for each event type. For example, interaction events logs information related to identification of the objects in the interaction.

The event model is the basis for the game data logging framework. It encapsulates the information that is desired by users and classifies events in three groups: immersion, quest, and social. The immersion group represents events related to immersion aspects of the game. The quest group represents events related to quest creation, execution, and analysis. Lastly, the social group represents events related to social interactions in the game.

Aside of classifying events in groups, they are also categorized into three different types: time events, interaction events, and emergent events. The time type represents events that are logged at specified time intervals and is parameterized by a time interval. Interaction type represents events related to interactions with other game objects and is parameterized by frequency. For example, log the attack event every third strike. The emergent type represents events that occur from internal state changes, such as dying from loss of health. With this event classification, the information to be logged can be customized for each group and type.

The data logging framework has four different log specification interfaces: In-situ, Aggregate, Programmer, and Player. The in-situ interface is an in-game interface to display the event log, allowing designers to log events via the game interface as they occur. Figure 2 illustrates the in-situ interface.

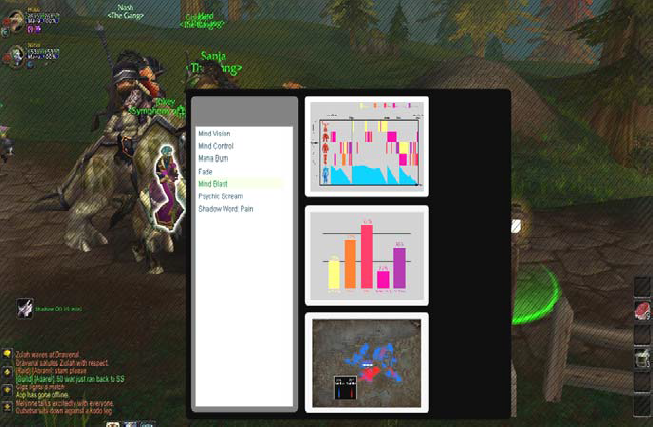


Figure 2: In-situ log specification interface. The designer can highlight game objects (white halo to the left of menu) and request a data log and visualization. Figure taken from (JOSLIN; BROWN; DRENNAN, 2007).

The aggregate interface summaries logged information by using graphs, facilitating the tracking of data-streams. The interface is illustrated in Figure 3. The programmer interface allows the programmer to specify data logging from the source code inside the programming IDE. The programmer interface is mainly used to log game data for debugging purposes. Lastly, the player interface allows players to provide feedback on various aspects of the game by reporting their experience. This interface is integrated with the game and periodically asks questions to the testers about their impressions on several aspects of the game, adding subjective impressions of fun experienced by the player in the log for future analysis.

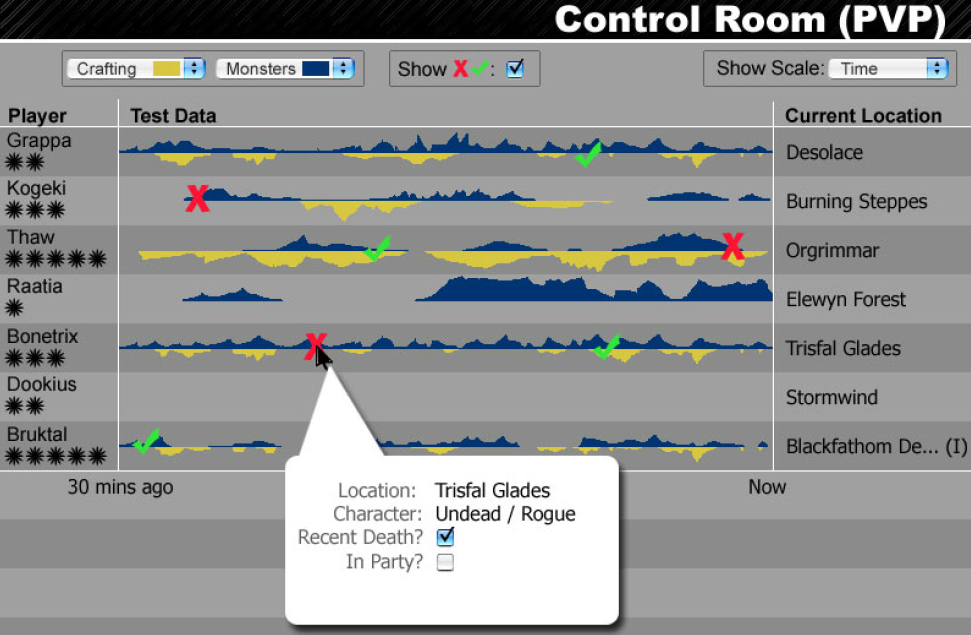


Figure 3: A mock-up example of the aggregate interface of a game as it is being played. Figure taken from (JOSLIN; BROWN; DRENNAN, 2007).

This data logging framework was mainly designed for online games to increase efficiency in gameplay verification process, to reduce testing expenses, and improve quality assurance. The framework address an overall data logging by using event models designed for logging, while also providing some basic data-stream visualization.

## Tracking Real-Time User Experience

The *Tracking Real-Time User Experience* (TRUE) approach (KIM *et al.*, 2008) combines human-computer interaction (HCI) instrumentation, which collects *user initiated events[[1]](#footnote-1)* (UIEs), and log file analysis techniques in order to automatically record user interactions with systems or games. While the focus of HCI instrumentation is to collect the frequency count of events, TRUE logs the sequences of events along with their timestamps. These sequences of events are important to understand user behavior. While typical HCI instrumentation logs how many times the user accessed the Help function from the system, TRUE logs the sequence of events that led the user to use the Help function for each occasion.

Another key aspect of the TRUE is the type of collected information. Instead of recording generic low-level events, such as mouse coordinates and function calls, TRUE collects event sets containing both the event as well as contextual information from the event. For example, in a game where the player died, TRUE records the player’s death, the equipment the player was using, the difficulty setting the player playing, the monster that killed the player (if that was the case), and other useful elements that might determine the cause of the player’s death.

The TRUE architecture is illustrated in Figure 4. The data capture occurs at real time while the user is using the system or playing the game. The data gathering collects system events and their contextual information, along with timestamps indicating when they occurred. At the same time, TRUE captures digital videos of user interaction with the system. The video is automatically synchronized with the event’s timestamps and indexed, allowing jumping to particular events relevant to the analysis. This link between event and video was stated by Kim (2008) to be an effective approach for understanding the users behaviors and how they interact with the systems. The last data capture from TRUE is in the form of a survey available to the user after finishing his interaction with the system. The survey is aimed at capturing information that might have been missed by the tracked UIEs. For example, when testing a game using the TRUE approach, a brief survey is displayed to the user asking the participant whether he enjoyed the game and how difficult it was. This type of survey is used to avoid making wrong inferences about the game by directly asking the player certain questions related to his game experience. For example, failing in a game may sometimes be a motivating part of the fun, while winning at the first attempt might indicate the game was too easy to the player.



Figure 4: The TRUE architecture. Figure taken from (KIM *et al.*, 2008).

The captured data is available to the viewer for analysis by visual representations in order to be easier to spot points of interest. The data visualization varies with the type of analysis. However, the general approach is the same: First, TRUE creates series of graphs and tables containing information needed to detect possible problems. Possible problems and data visualization are previously defined (customized) by designers when integrating TRUE in the system. Then, TRUE generate graphs and tables to answer specific questions according to previously established goals. Figure 5 shows different examples of data visualization.



Figure 5: Examples of data visualization from TRUE, customized by designers during its integration with the application. A graph showing the average player death for each mission in a game (a) and which race was selected by players (b). A map displaying death locations in a Real Time Strategy game (c). A visualization of where users clicked in the spreadsheet (d). Figures taken from (KIM *et al.*, 2008).

The TRUE approach is an approach for videogame industry designed to detect issues and understand the causes the same way a usability testing does. It also incorporates attitudinal behavior by using surveys, aiding in the understanding the player’s emotional experience. TRUE can also be used to understand how users utilize products (i.e., systems, applications, or tools). However, its common usage is for beta testing games, making observations of usage and to understand how people interact and play games.

## Playtracer

Playtracer (ANDERSEN *et al.*, 2010; LIU *et al.*, 2011) is a visual tool designed to illustrate how groups of players move through the game space. Playtracer can be used for behavior analysis in games with the concept of state transitions. The transitions in the game are represented as game states by applying the Classical Multidimensional Scaling (CMDS) (COX; COX, 2010) to project the game space in two dimensions. Thus, Playtracer aids the designer by showing common pathways and alternatives that players used to succeed or fail in their tasks, identifying pitfalls and anomalies in the scene. It also tracks how players progressed through the levels in the game.

In Playtracer, a play trace is the path that each player took in the game, scaled to two dimensions by using CMDS. The transformation places similar states close to each other while dissimilar states are placed apart. Thus, CMDS allows for easy similarity identification between states that were visited by players. The distance between states is calculated by following specific metrics that are customized by the game. Distance metrics are also used to analyze different features of the game. For example, a distance metric with a component to compare how many steps are necessary to reach a goal state will cluster goal states while placing states that are difficult to reach the goal far away. Thus, the designer can identify players that are not making progress in their goals and possibly investigate the issue.

An example of the Playtracer graph output is illustrated in Figure 6. The input for Playtracer is a list of all states that the players visited during the game and a distance metric to calculate the distance between states. The output is a directed graph where the vertices represent the game states and the directed edges are the movements the player did to move from one state to another. Furthermore, the size of the vertex, or state, is proportional to the number of players that reached that state. Thus, the size of the vertex can be used to identify which states were more visited by players.



Figure 6: Playtracer state visualization. Circles represent game states and edges represent paths taken by players. The distance metric used for this graph clustered states with similar pieces of statements. Figure taken from (LIU *et al.*, 2011).

Moreover, the graph utilizes color to distinguish displayed information. A yellow state is the game’s initial state and green state represents the goal. Blue edges represent moves made by players who won the game and red edges for those who lost. The shades between red and blue represent the probability that the player who reached the state completed the game successfully. Lastly, cycles in the graph represent failed attempts from the players, where they made a move that returned to a previous state.

The main focus of the Playertracer is to display aggregated user behavior in a graph in order to aid in understanding common strategies adopted by players and to identify points of confusion for players. To solve problems related to game with many states, Playtracer uses features to aggressively cluster states together to make a cleaner visualization. Another feature is to make equivalent states to be represented by the same state, reducing the number of states displayed in the screen. Lastly, it is possible to filter the graph (winners from losers) to visually compare their respective behaviors in order to identify similarities. A drawback is that Playtracer does not take in consideration temporal information. For example, Playtracer does not say when each state was visited by each player, only that they visited it while playing the game.

## Play-Graph

Play-Graph (WALLNER; KRIGLSTEIN, 2012; WALLNER, 2013) is a recent concept to formally describe and visualize gameplay data by using different graph visualizations to describe multiple variables and their interrelations along with the temporal progression of players. The gameplay analysis of the play-graph illustrates the sequence of states performed by actions from the players over the course of the game. In the Play-Graph context, a game state describes a certain configuration of the game or an entity, while actions are player interactions within the game, like shooting, jumping, or using an object. These actions are responsible for changing the current game state due to influences generated in the current state or to other entities.

In this concept, a game is viewed as a finite state machine with a finite number of states and transitions between them. Thus, the state machine can be represented by a directed graph with each vertex representing a state from the game and edges representing actions. States are composed of a set of attributes from the game. Actions are triggered by players at a specific point in the game and can be of different types or have a duration. For example, possible types of actions are: running, walking, jumping, and pulling a lever. Furthermore, actions (edges) linked to states (vertices) can have labels to provide additional information to differentiate from other states and actions.

The Play-Graph visualization is composed of Node-link diagrams. Nodes, or vertices, in the graph represent game states. The size of each node is directly related to the number of players that visited that state at any time during the game. Moreover, multiple edges from the same source to the same target are merged together to create a meta-edge. The thickness of each meta-edge is proportional to the number of edges that composes the meta-edge. It is possible to have two meta-edges between two nodes due to the nature of the directed graph, where each meta-edge represents a different direction. Furthermore, each node and edge type in the graph is distinguished by colors. Lastly, icons in the graph represent players in the game. The icon color is directly related to certain attributes from the player (gender, age, character class). Figure 7 illustrates the basic representations from the graph.



Figure 7: Basic elements from the Play-Graph: nodes (a) representing states, directed edges (b) representing player’s actions, meta-edges (c), and player icon (d) representing time-dependent location of individual players. Figure taken from (WALLNER, 2013).

Viewing gameplay data by graphs allows for the usage of graph theory concepts to study and understand player behaviors. However, due to the nature of how the graph is structured in Play-Graph (limiting vertices as states and edges as actions), understanding player behaviors may be limited by the player progression in the game (i.e., killed a boss), and not by how he interacted with the world (i.e., combat rounds from the battle against the boss). Apparently, there is no way to determine interactions or influences. Only the changes from one state to another, caused by an action executed by the player, can be identified. Thus the graph illustrates the player’s progression in the game, demonstrating actions he made to change states. However, influences in the player’s action, such as an influence from an NPC, that affected the transition of one state to another are not present in the graph (there are no edges linking edges). Besides, this visualization was not designed to track individual progression but to track the player population flow.

## Final Considerations

This chapter presented existing approaches for gameplay data logging and visualization. However, such approaches are aimed at players’ behaviors, game balancing, and game testing by identifying issues. Furthermore, these approaches use graphs to illustrate the general population behavior in order to be used by designers to improve the game.

The approach we are proposing in this dissertation is to follow the player’s actions more closely. To state his interactions in the game with other game objects and what influences affected each action. This allows for understanding the consequences of each action and how they affected the outcome of the game. In other words, the provenance of the player’s story, how it was generated, which components were present in the story, and how these components interacted with the player. To known how each action affected the story progression and how the player was affected by the actions of others. Thus allowing for a deeper understanding of how the story progressed by using a provenance graph representing the data gathered during the game session.

Provenance is a well known and understood term used in arts to describe an object’s life cycle. However, provenance also began to be used for digital information. The next chapter describes the two existing digital provenance models, stating their characteristics and features, while chapter 4 describes our proposed approach based on provenance.

## References

AMBINDER, Mike. *Valve’s approach to playtesting: The application of empiricism.* . Game Developer Conference: [s.n.]. , 2009

ANDERSEN, Erik *et al.* Gameplay analysis through state projection. *Proceedings of the 5th International Conference on the Foundations of Digital Games*, FDG ’10. p. 1–8, 2010. Accessed: 14 set. 2012.

COX, Trevor; COX, Michael. *Multidimensional Scaling, Second Edition*. United Kingdom: CRC Press, 2010.

DEROSA, Phillip. *Tracking Player Feedback To Improve Game Design*. Gamasutra. Available: <http://www.gamasutra.com/view/feature/129969/tracking\_player\_feedback\_to\_.php>. Accessed: 28 jun. 2013.

DIXIT, Priyesh; YOUNGBLOOD, Michael. *Understanding playtest data through visual data mining in interactive 3D environments.* . [S.l.]: The University of North Carolina at Charlotte, 2008.

FULLERTON, Tracy; SWAIN, Christopher. *Game Design Workshop: A playcentric approach to creating innovative games*. Amsterdam: Morgan Kaufmann/Elsevier, 2008.

HOOBLER, Nate; HUMPHREYS, Greg; AGRAWALA, Maneesh. Visualizing Competitive Behaviors in Multi-User Virtual Environments. VIS ’04, 2004, Washington, DC, USA. *Anais*... Washington, DC, USA: IEEE Computer Society, 2004. p. 163–170. Available: <http://dx.doi.org/10.1109/VISUAL.2004.120>. Accessed: 28 jun. 2013.

JOSLIN, Simon; BROWN, Ross; DRENNAN, Penny. The gameplay visualization manifesto: a framework for logging and visualization of online gameplay data. *Comput. Entertain.*, v. 5, n. 3, jul. 2007. Available: <http://doi.acm.org/10.1145/1316511.1316517>. Accessed: 27 jun. 2013.

KIM, Jun H. *et al.* Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’08. p. 443–452, 2008. Accessed: 26 jun. 2013.

LIU, Yun-En *et al.* Feature-based projections for effective playtrace analysis. *Proceedings of the 6th International Conference on Foundations of Digital Games*, FDG ’11. p. 69–76, 2011. Accessed: 24 jun. 2013.

ROMERO, Ramon. *Tracking attitudes and behviors to improve games*. . Game Developer Conference: [s.n.]. , 2008

THOMPSON, Clive. *Halo 3: How Microsoft Labs Invented a New Science of Play*. [S.l: s.n.], 2007. Available: <http://www.wired.com/gaming/virtualworlds/magazine/15-09/ff\_halo>.

WALLNER, Günter. Play-Graph: A Methodology and Visualization Approach for the Analysis of Gameplay Data. *Proceedings of the 8th International Conference on the Foundations of Digital Games*, FDG ’13. p. 253–260, 2013.

WALLNER, Günter; KRIGLSTEIN, Simone. A spatiotemporal visualization approach for the analysis of gameplay data. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’12. p. 1115–1124, 2012. Accessed: 24 jun. 2013.

ZOELLER, Georg. *Development telemetry in video games projects*. Available: <http://gdc.gulbsoft.org/talk>. Accessed: 28 jun. 2013.

1. According to KIM (2008), UIEs are “events that occurred when the user interacted with the system”. For example, the dragon’s death sequence begins after the player killed the dragon, where the death sequence is the event that occurred by the player’s action. [↑](#footnote-ref-1)