# – Game Flow Analysis

## Introduction

## 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, which describes which attributes and information are logged for each type of event. For example, interaction events will log information related to the identification of the object interaction.

The event model is the basis for the framework. It encapsulates the information that is desired by the 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 is represents events related to quest creation, execution, and analysis. Lastly, the social group represents events related to social interactions and factors in the game.

Aside of being classified in groups, events are also categorized into three different types: Time events, Interaction events, and Emergent events. The time type is events that are logged at specified time intervals. This type is parameterized by a time interval. Interaction type is events related to interactions with other game objects and is parameterized by frequency. For example, log an attack event every third strike. The emergent type is an event that occurs from internal state changes, such as dying from loss of health. Thus, the information to be logged is customized for each group and type of event, similar to a typical IDE debugging for generating run-time data logs.

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 1 illustrates the in-situ interface.

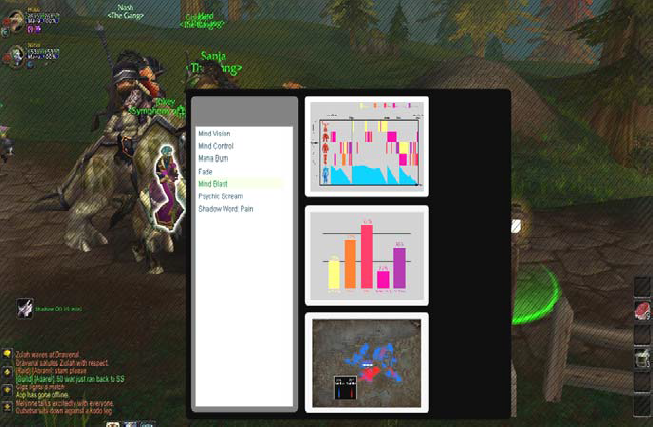


Figure 1: In-situ log specification interface. The designer can highlight game objects (white halo to the left of menu) and request a data log and visualization. Figure taken from (JOSLIN; BROWN; DRENNAN, 2007).

The aggregate interface summaries logged information by using graphs, as illustrated by the mock-up example in Figure 2. The programmer interface allows the programmer to specify data log from the source code inside the programming IDE. This interface is mainly used to log game data for debugging purposes. The last interface is the player interface, which allows players to provide feedback on various aspects of the game by reporting their experience. This interface is integrated with the game by asking 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.

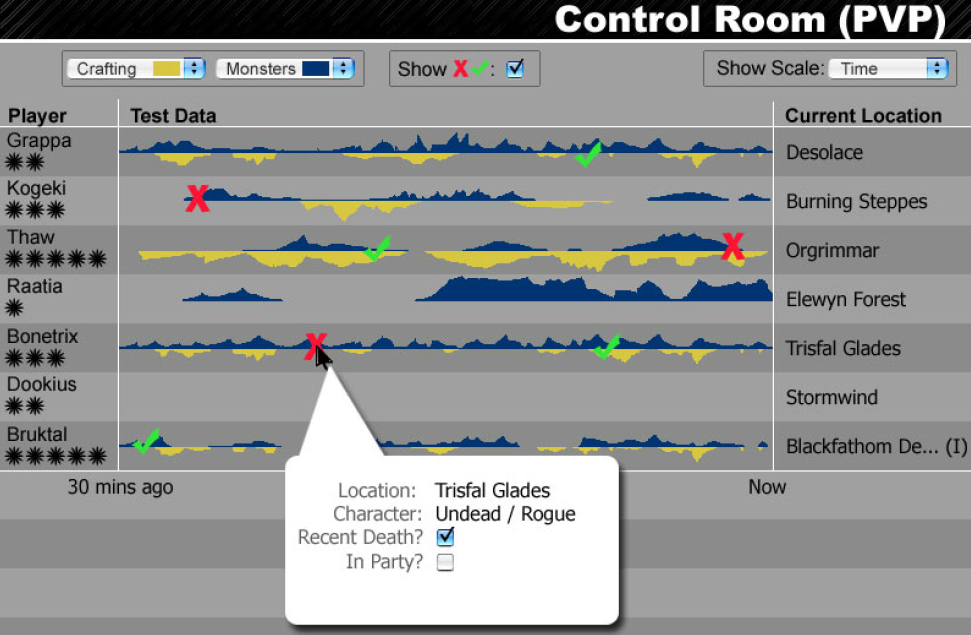


Figure 2: A mock-up example of the aggregate interface of a game as it is being played. Figure taken from (JOSLIN; BROWN; DRENNAN, 2007).

## 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 logfile 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, the TRUE system logs the sequences of events along with their time stamps. These sequences of events are important to understand the user behavior. While typical HCI instrumentation will log how many times the user accessed the Help function from the system, TRUE will log 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 information collected. Instead of recording generic low-level events, such as mouse coordinates and function calls, TRUE collect 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 death, the equipment the player was using, the difficulty setting the player was using, the monster that killed the player (if that was the case), and other elements that might be useful to determine the cause of the player’s death.

The TRUE architecture is illustrated in Figure 1. The data capture occurs at real time while the user is using the system or playing the game. The data collect is related to system events and their contextual information, along with timestamps indicating when they occurred. At the same time, TRUE captures digital videos of the users interacting with the system. The captured video is automatically synchronized with the timestamps and indexed to the events, allowing jumps 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 after the user finished 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 needing to be fixed. Failing in the game may sometimes be a motivating part of the fun, while winning at the first attempt might indicate the game was too easy.



Figure 3: The TRUE architecture. Figure taken from (KIM *et al.*, 2008).

The captured data is available to the viewer for analysis by visualizing and transforming the data in order to make it easier to spot points of interest and their causes. The data visualization varies with the type of analysis. However, the general approach is the same: First, TRUE creates a series of graphs and tables containing the information needed to detect possible problems. These types of problems, and the data visualization, are previously defined (customized) by designers during the integration of TRUE with the system. Then, TRUE builds graphs and tables to answer specific questions according to previously established goals. Figure 2 shows different examples of data visualization.



Figure 4: 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 deaths 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).

Thus, TRUE is an approach for videogame industry designed to detect issues and to understand the root causes the same way usability testing does. It also incorporates attitudinal behavior by using surveys, aiding in the understanding of the player’s emotional experience. TRUE can also be used to understand how users utilize products (i.e. systems, applications, or tools). Its common usage is for beta tests of products (games and software), making unobtrusive observations of usage. However, TRUE was primarily developed 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 from 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 while also tracking how particular 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 easily identify similarity between states that were visited by players. The distance between states is calculated by following specific metrics that are customized by each 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 result in clustering 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 at Figure 3. The input for Playtracer is a list of all the states that the player visited during the game and a distance metric to calculate the distance between states and generates the graph. In the Playtracer’s graph, 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 5: 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 but 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 do not take in consideration temporal information. For example, Playtracer do not measure how long the player took to execute an action or the time between actions. Long pauses could mean that the player was thinking his next course of action. Likewise, consecutive actions could imply frustration or panicking, which is a common behavior in games.

## Play-Graph

Play-Graph (WALLNER; KRIGLSTEIN, 2012; WALLNER, 2013) is a concept to formally describe and visualize gameplay data by using different graph visualizations to describe changes between two datasets. 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 and each vertex in the graph represents a state from the game while edges represent actions. States are composed of a set of attributes that are used to define a state from the game. Meanwhile, 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. The possible types are defined by the game. 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 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 different colors. Lastly, icons in the graph represent the players in the game. The icon color is directly related to certain attributes from the player (gender, age, character class). Figure 4 illustrates the basic representations from the graph.



Figure 6: Basic elements from the Play-Graph: (a) nodes representing states, (b) directed edges representing player’s actions, (c) meta-edges, and (d) player icon representing time-dependent location of individual players. Figure taken from (WALLNER, 2013).

## Final Considerations

## References

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. Acesso em: 14 set. 2012.

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

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. Disponível em: <http://doi.acm.org/10.1145/1316511.1316517>. Acesso em: 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. Acesso em: 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. Acesso em: 24 jun. 2013.

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. Acesso em: 24 jun. 2013.

1. From (KIM *et al.*, 2008), UIEs are events 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 occurred by the player’s action. [↑](#footnote-ref-1)