Troy Costa Kohwalter

PROVENANCE IN GAMES

Thesis presented to the Computing Graduate program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Topic area: Visual Computing.

Advisors: Prof. D.Sc. Esteban Gonzalez Walter Clua
Prof. D.Sc. Leonardo Gresta Paulino Murta

Niterói
2013

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

K79 Kohwalter, Troy Costa
Provenance in games / Troy Costa Kohwalter. – Niterói, RJ : [s.n.], 2013.
106 f.

Dissertação (Mestrado em Computação) - Universidade Federal Fluminense, 2013.
Orientadores: Esteban Gonzalez Walter Clua, Leonardo Gresta Paulino Murta.

1. Computação visual. 2. Jogo por computador. 3. Grafo. I. Título.

CDD 006.6

TROY COSTA KOHWALTER

PROVENANCE IN GAMES

Thesis presented to the Computing Graduate program of the Universidade Federal Fluminense in partial fulfillment of the requirements for the degree of Master of Science. Topic area: Visual Computing.

Approved on August 2013.

APPROVED BY

_____

Prof. D.Sc. Esteban Gonzalez Walter Clua – Advisor

IC-UFF

_____

Prof. D.Sc. Leonardo Gresta Paulino Murta – Co-Advisor

IC-UFF

_____

Prof. D.Sc. Daniel Cardoso Moraes de Oliveira

IC-UFF

_____

Prof. D.Sc. Flávio Soares Correa da Silva

IME-USP

Niterói

2013

For Kira, my faithful and beloved companion.

ACKNOWLEDGMENTS

I would like to thank my parents and brother for supporting my decisions.

I am grateful to my cousins, aunts, uncles, and grandparents for providing me wonderful moments.

This thesis would not have been possible without the help, patience, and counsels of my advisors, Esteban and Leonardo.

I want to thank my fellow postgraduate students in the computer science department for promoting a stimulating and welcoming academic and social environment.

I would like to acknowledge the financial, academic and technical support of the Universidade Federal Fluminense, CNPq, FAPERJ, and CAPES.

And lastly, my deepest appreciation goes to Kira, to whom I dedicate this entire thesis.

RESUMO

Ganhar ou perder uma sessão de jogo é a consequência final de uma série de decisões e ações feitas durante o jogo. A análise e compreensão dos eventos, erros, e os fluxos de um jogo concreto pode ser útil por diversos motivos: compreender os problemas de jogabilidade, buscar dados de situações específicas e até mesmo entender os aspectos educacionais em jogos sérios. Erros cometidos pelos jogadores podem resultar em falha para completar os objetivos do jogo. Estes erros, muitas vezes difíceis de serem detectados ou reproduzidos em sessões subsequentes, prejudicam diretamente a capacidade de aprendizagem dos jogos sérios ou comprometem aspectos lúdicos de um jogo. Para resolver esta questão, apresentamos uma nova abordagem baseada em conceitos de proveniência a fim de modelar e representar um fluxo de jogo. Primeiramente, propomos um processo de modelar e mapear os dados do jogo para o conceito de proveniência, a fim de gerar um grafo proveniência que é utilizado para posterior análise. Como prova de conceito, a abordagem proposta e a geração do grafo foram aplicadas em um jogo de Engenharia de *Software*, permitindo usuários identificar erros cometidos pelo jogador através de analises no grafo de proveniência gerado a partir de dados coletados do jogo.

Palavras-chave: fluxo de jogo, análise, comportamento de jogadores, registro de dados, jogabilidade, proveniência, grafo.

ABSTRACT

Winning or losing a game session is the final consequence of a series of decisions and actions made during the game. The analysis and understanding of events, mistakes, and flows of a concrete game play may be useful for different reasons: understanding problems of gameplay, search data of specific situations, and even understanding educational aspects in serious games. Mistakes made by players may result in failure to complete the game objectives. These mistakes, which are often difficult to detect or to reproduce in subsequent trials, directly jeopardize the learning capabilities of serious games or compromise entertaining aspects of the game. In order to solve this issue we introduce a novel approach based on provenance concepts to model and represent a game flux. Firstly, we propose a process of modeling and mapping the game data to match the provenance concepts, in order to generate a provenance graph to be used for further analysis. As a proof of concept, we also applied our proposed framework and graph generation in a Software Engineering game, allowing users to identify player's mistakes by analyzing the generated provenance graph from collected gameplay data.

Keywords: Game flux, analysis, player behavior, data logging, gameplay, provenance, graph.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

CMDS – Classical Multidimensional Scaling

GUI – Graphical User Interface

GVM – Gameplay Visualization Manifesto

HCI – Human-Computer Interaction

HP – Hit Points

IPAW – International Provenance and Annotation Workshop

NPC – Non Player Character

OPM – Open Provenance Model

RPG – Role-Playing Game

SDM – *Software Development Manager* computer game

TRUE – Tracking Real-Time User Experience

UIE – User Initiated Events

TABLE OF CONTENTS

CHAPTER 1 – INTRODUCTION

1.1 MOTIVATION

During a game session, the player faces many challenges that require decisions and actions in order to overcome them. The fun factor of a game arises from these elements, which must be well calibrated and balanced, according with the player profile. In many situations, analyzing and understanding the events, mistakes, and flows of a concrete gameplay[1] experience, known as game flux analysis, is useful for understanding the achieved results, crash locations, goal violations, and movement patterns during the session (ZOELLER, 2010). Game flux analysis is also fundamental for detecting symptoms of problems that occurred due to wrong decision-making or even bad gameplay and game balance design (DANKOFF, 2011; ZOELLER, 2010). One way or another, the current analysis methods can be subjective to the assumptions from the designer related to player behavior (DIXIT; YOUNGBLOOD, 2008). Furthermore, depending on the game dynamics and its complexity, it would require playing the game successively by making the same decisions to intuitively guess which ones were responsible for generating the observed outcomes. Therefore, reproducing the same state can be unviable due to the difficulty of generating the same game state in order to identify, in a trial and error approach, the source of a problem. In addition, examining the game flux allows the identification of good and bad attitudes made by players.

Game flux analysis techniques can be divided in two phases (JOSLIN *et al.*, 2007; KIM *et al.*, 2008): data logging and visualization. The data logging phase is responsible for collecting gameplay information during a game session. The visualization phase, which recently began to be used by the game industry, displays the gathered gameplay information in the form of graphics and graphs for analysis. Developers and game designers use these graphs and graphics to study certain aspects of the game, providing insights about player behavior or game issues.

Therefore, almost all AAA[2] game titles have some form of game development telemetry due to the importance of game flux analysis (ZOELLER, 2010). According to ZOELLER[3] (2010), game development telemetry is an "*automatic measurement and*

[1] Gameplay is defined as "the total experience provided by a game's structure and mechanics" (THOMPSON, JIM *et al.*, 2007).

[2] AAA or triple-A game is a game developed by a large studio and funded by massive budget (SCHULTZ, 2006).

[3] Georg Zoeller is the Lead Technical Designer for BioWare Austin.

transmission of data from game executable, build pipeline and development tools for recording, analysis, and workflow improvement". Game development telemetries are mainly used by the game industry to analyze gameplay data to understand the customer's experiences in the game, to identify post launch issues, and to understand the market for future games releases (ZOELLER, 2010).

The known approaches are developer-oriented, which means that the game analysis is done by developers to improve their games. However, we believe that players can also directly benefit from a game flux analysis to understand how the game reacted to their actions and decisions. This might be the case of hardcore players, which tries to find ways to min-max the game to create the best character, or by game *modders*, which are players that develop game content by using the tools provided by the game company. Existing approaches for this purpose, such as the replay of a game session, only show explicit influences. For example, in a replay of a racing game, it is possible to identify only shallow reasons, such as that the car rolled over from tight turns or collided with a fence. However, other deeper reasons may involve the car's tire pressure or suspension tuning, which can be customized by the player. These other possible reasons are difficult to visually identify when watching a replay and can be unnoticeable to less experienced players.

Therefore, this work main motivation is to facilitate user's understanding about how events emerged during the game session and how each action influenced the outcomes. This knowledge can be used in future game sessions to avoid making the same mistakes or even to adjust gameplay features.

1.2 GOALS

Given the aforementioned motivation, the aim of this work is to present a new approach for game flux analysis. The goal is to improve the understanding of the game flux, providing insights on how the story progressed and the influences on the outcomes. In order to improve understanding, this work provides the means for analyzing the game flux by using provenance[4]. Provenance analysis is done by processing collected gameplay data and generating a provenance graph, which relates the actions and events that occurred during the game session. This provenance graph allows identifying critical actions that influenced the game outcome and helps to understand how events were generated and which decisions

[4] In this context, Provenance refers to the documented history of an object's life cycle and is generally used in the context of art, digital data, and science (PREMIS WORKING GROUP, 2005).

influenced them. This process may also aid in the identification of mistakes, allowing the player to reflect upon them for future interactions.

Thus, this work proposes a conceptual framework that collects information during a game session and maps it to provenance terms, using digital provenance (FREIRE *et al.*, 2008) concepts for representing the game flux. The conceptual framework provides the means for game flux analysis by using and manipulating a provenance graph that represent the gathered game information.

This work also aims at applying the proposed conceptual framework in a serious game in order to improve the player's understanding of the lessons taught by the game. Understanding the reasons that lead to the obtained outcome might aid the player to avoid making further mistakes and aid in the assimilation of the knowledge taught by the game.

Even though the scenario used in this work is over a serious game, we believe that the concepts discussed here are applicable to any kind of game and are useful to support advanced game flux analysis, such as gameplay design, gameplay balancing, gameplay metrics, data mining, and even for storytelling.

1.3 RESEARCH QUESTIONS

Our approach and its evaluation have as main objective to answer the following research questions:

- Does provenance analysis help to understand events that emerged during a game session?
- Is provenance analysis faster than only watching a replay of the game session?
- Is provenance analysis more accurate than only watching a replay of the game session?

1.4 CONTRIBUTIONS

This work introduces new perspectives on game flux representation and analysis that can consolidate the knowledge gathered during a game session. This knowledge can help on confirming the hypotheses formulated by players on how events affected the game, supporting tutors for a better guidance when applied in a serious game, and extracting behavior patterns from individual sessions or groups of sessions. The provenance gathering also opens new research possibilities for behavior pattern data mining, provenance in storytelling, detecting gameplay design issues, gameplay metrics, and automatic gameplay refinement and balance.

The provenance visualization can occur both on-the-fly or in post-mortem sessions. It allows for the discovery of issues that contributed to specific game fluxes and results achieved throughout the game session. This analysis can be used on games to improve understanding of the game flux and identifying actions that influenced the outcome, aiding to understand why they happened the way they did. It can also be used to analyze a game story development, how it was generated, and which events affected it.

1.5 ORGANIZATION

This work is organized in six other chapters, beside this introduction. Chapter 2 outlines the related work for this work. It presents existing usages of game flux analysis, ranging from gameplay data logging to game analysis in the game industry. It also describes known approaches for gameplay data logging and gameplay data visualization.

Chapter 3 outlines part of the necessary knowledge base for this work. It describes concepts of provenance in order to gather historical information about objects for further analysis. It also presents the existing provenance models (OPM and PROV) that can be used for provenance of digital information. Lastly, it presents a comparison between models, pointing out their similarities.

Chapter 4 presents the conceptual framework, denominated as *Provenance in Games*. This chapter describes how the gameplay data is gathered, structured and mapped to be used in a provenance graph. Then, it outlines rules to interpret the gathered data for the generation of the provenance graph. Lastly, it describes some features to distinguish information in the graph and to aid in the analysis.

Chapter 5 presents the materialization of the conceptual framework presented at Chapter 4, encompassing both the collection and visualization phases. It provides an overview of a serious game named SDM, previously developed by the author, describing the game and how the gameplay data is gathered. It also outlines implementation details for the provenance graph visualization tool developed in this work. Furthermore, it introduces a guiding example, which is the same game session used at Chapter 6 during the experiment. Lastly, it shows details about generating the provenance graph from the gathered information, graph representations, and analysis features available at the developed tool over the guiding example.

After describing the approach and the graph visualization tool, Chapter 6, describes the evaluation performed on the usage of provenance analysis to understand game events. The planning and execution of the experiments are detailed, indicating how we obtained and

processed the data for the assessment. The results are analyzed through hypothesis testing. Finally, it presents some threats for the validity of the experiments.

Finally, Chapter 7 concludes this work, listing contributions, limitations, and future work.

CHAPTER 2 – GAME FLUX ANALYSIS

2.1 INTRODUCTION

Methods for automatic logging of gameplay data have become an important component of game design in the last few years (WALLNER, 2013). The collected data can be used for different purposes, such as behavior analysis, identifying strategies, detecting bugs, testing games, balancing the game experience, classifying users, and even understanding common behaviors. However, the analysis of the collected data can be challenging due a huge amount of generated data. Thus, data visualizations have become a promising solution for exploring and understanding game fluxes.

Visual data mining approach is useful when the designer is not familiar with the gathered data or has vague analysis goals (LIU *et al.*, 2011). Therefore, visualization methods are gaining popularity among designers for understanding gameplay data in the game industry (WALLNER, 2013). For example, a common visualization method is heat map (DRACHEN; CANOSSA, 2009), which uses colors in a two-dimensional map to reflect density of certain variables in particular locations of the game. Recently, *BioWare, a Canadian video game developer,* used heat maps to analyze common bug locations (ZOELLER, 2010), while *Valve* used heat maps to analyze multiplayer maps in Team Fortress 2 (AMBINDER, 2009). Meanwhile, *Bungie* and *Microsoft* used heat maps to determine common places where players died in *Halo 3* (ROMERO, 2008; THOMPSON, CLIVE, 2007). Figure 1 illustrates an example of heat map usage in the *Unreal Engine* to show the locations in the map with kill events by using a specific weapon, which gives the idea of map coverage. The colors in the heat map range from purple (almost no activity) to red (highest activity). Another approach similar to heat maps was proposed by NASCIMENTO (2010) , where it renders trail lines in the game map to represent paths taken by characters in the environment.

Another common usage for game data logging and visualization in the game industry is aiding during validation 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 playtesting to determine where players spent their time and which special powers were used. Other researchers also tried to analyze movements during battles (HOOBLER *et al.*, 2004) and identify player behaviors (DIXIT; YOUNGBLOOD, 2008).

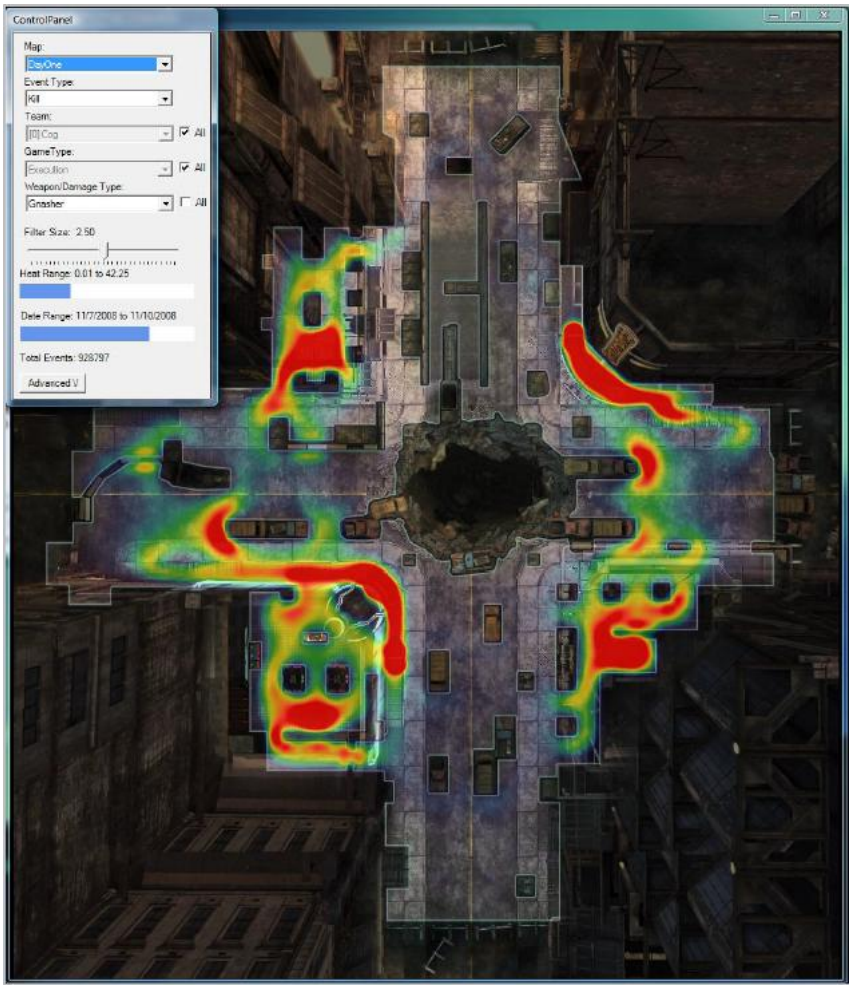

Figure 1: A heat map representing shotgun kills on Gears of Wars 2.

This chapter describes some approaches related to game flux analysis, outlining techniques for data logging and data visualizations. The criterion used for selecting the approaches is similar to snowballing sample (GOODMAN, 1961). The sampling procedure starts with a finite individual population as seed. Each seed in the sample is asked to name different individuals in the population. These new named individuals, who form the second stage, are asked to name more individuals, forming the third stage.

For the seed, we used the Play-Graph, a recent research that displays gameplay data by using graphs (WALLNER, 2013). From the seed we obtained another graph based visualization (WALLNER; KRIGLSTEIN, 2012) and an approach that combines behavioral and contextual data visualization (KIM *et al.*, 2008). From these, we selected another approach for understanding player behavior (DIXIT; YOUNGBLOOD, 2008), however we do not describe it on this chapter because it focus on visual representations of movement behavior (*i.e.* walking, jumping, and pirouettes) to overcome obstacles in the environment.

Nevertheless, from that approach we selected the last one, which proposes a framework for gameplay data logging (JOSLIN *et al.*, 2007).

This chapter is organized as follows: Section 2.2 describes a gameplay data logging framework proposed by Microsoft. Section 2.3 describes an approach for player behavior analysis using contextual data. Section 2.4 and 2.5 describes two visual tools designed to visualize gameplay data by using graphs. Lastly, Section 2.7 presents the final considerations of this chapter.

2.2 GAMEPLAY VISUALIZATION MANIFESTO

Joslin (2007) proposed the *Gameplay Visualization Manifesto* (GVM), which is a framework for gameplay data logging that uncovers gameplay events by attaching logging methods in game objects responsible for generating relevant 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 increasing the player's sensation of being involved in the game flux. The quest group represents events related to quest creation, execution, and analysis. Lastly, the social group represents events related to social factors in the game, such as group meeting or interaction with other characters.

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, such as logging 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. The aggregate interface summaries logged information by using graphs, facilitating the tracking of data-streams. 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 for 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.

This data logging framework was mainly designed for online games in order to increase efficiency in gameplay verification process, 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.

However, the main application is for collecting game metrics, such as player deaths, position, time spent in available features (*i.e.* crafting and fighting), item usage (*i.e.* equipment), actions performed, and player enjoyment. Furthermore, there is no documentation related to contextual information that can be used to understand player behavior.

2.3 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*[5] (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 the 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 game's difficulty setting set by the player, the enemy that killed the player (if that was the case), and other useful elements that might determine the cause of the player's death.

TRUE architecture is illustrated in Figure 2. The data capture occurs at real time while the user is using the system or playing the game. The data capture collects system events and

[5] According to KIM (2008), UIEs are "*events that occurred when the user interacted with the system*".

their contextual information, along with timestamps indicating when they occurred. At the same time, TRUE captures digital videos of user's screen, which shows the 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 for 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 player 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 for the player.

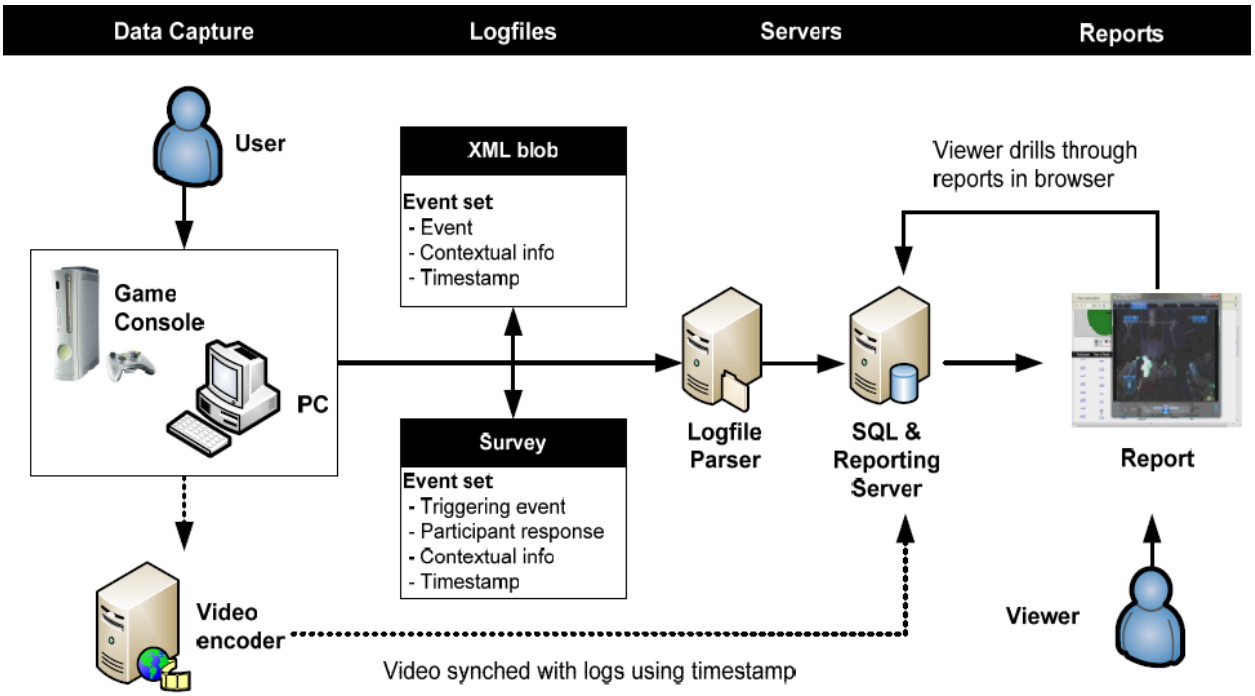

Figure 2: The TRUE architecture. Figure taken from KIM *et al.* (2008).

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. Figure 3 shows different examples of data visualization, customized by designers for the application. In the figure, there is a graph showing the average player death for each mission in a game (a) and which race was selected by players (b). Another possible

visualization is by using an in-game map to display death locations in a Real Time Strategy game (c) or by marking in the application where users clicked, such as in a spreadsheet (d).

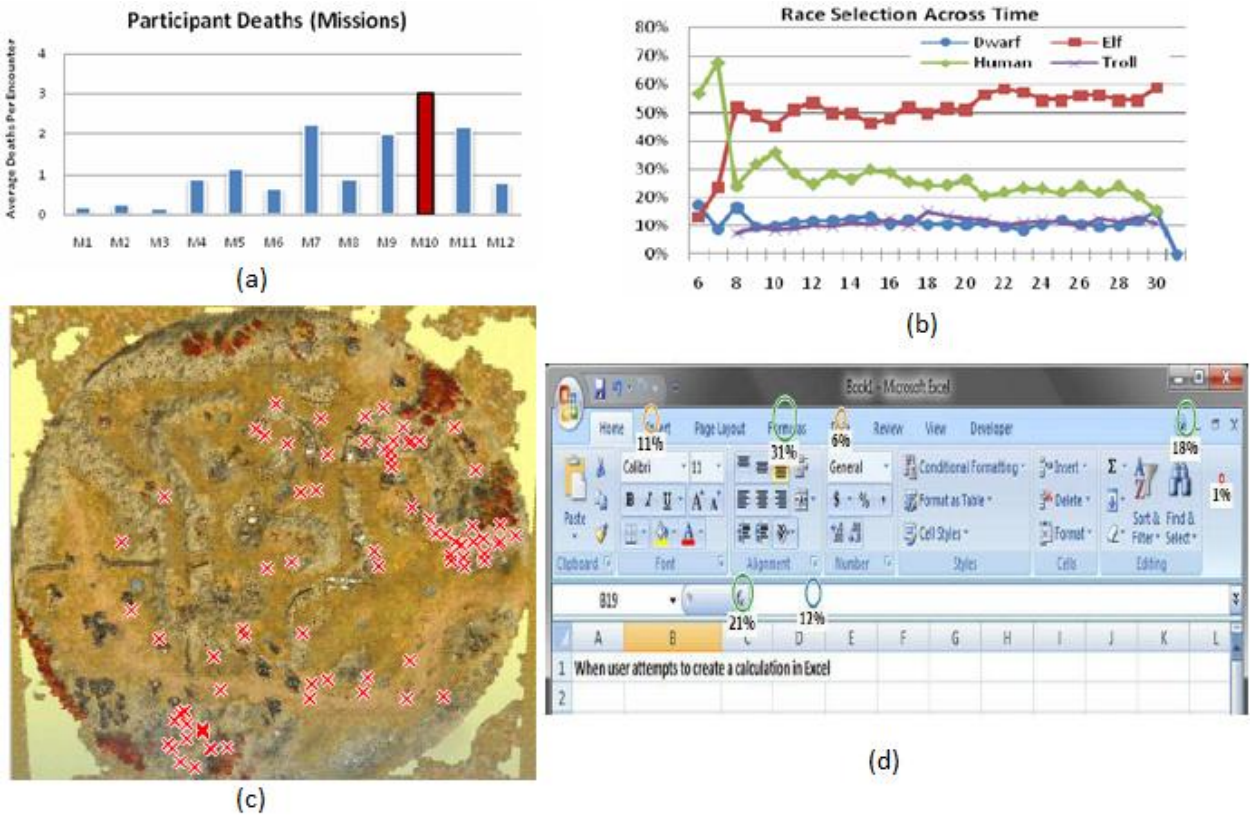

Figure 3: Examples of data visualization from TRUE. Figures taken from KIM *et al.* (2008).

The TRUE proposal 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 the understanding of the player's emotional experience. TRUE can also be used to understand how users interact with products (i.e., systems, applications, or tools). However, its common usage is focused at beta testing stages, making observations of usage and to understand how people interact and play games.

2.4 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 this system, 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.

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.

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 are 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.

As can be observed in Figure 4, most players moved from the initial state, at the center of the figure in yellow, to a purple state where most of the difficulties began, moving players to several different states further away from their goals. By also observing the figure, the goal state, in green, could only be accessed by two different states from the game, with one of them linking the purple state.

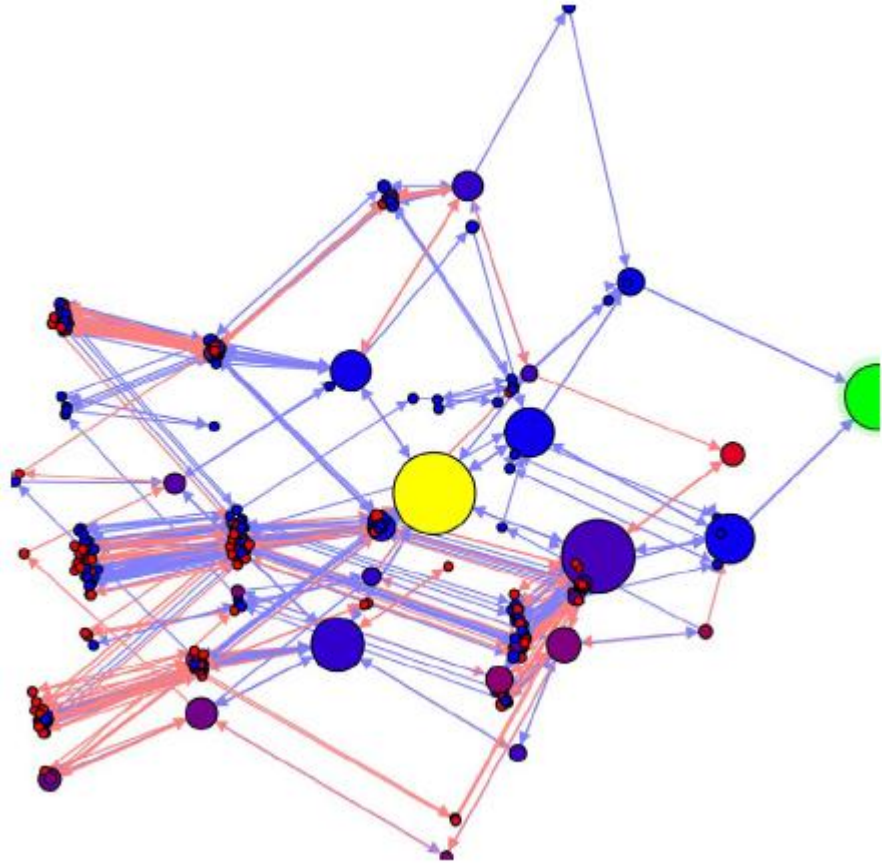

Figure 4: Playtracer state visualization. Figure taken from LIU *et al.* (2011).

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. The temporal information would allow stating the order of events in the game, shedding more light in the player's behavior. For example, Playtracer does not say when each state was visited by each player or the order they were visited, only that they visited it while playing the game.

2.5 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 consist on 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. The colors are chosen by the user in order to adapt the visualization to his needs. 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 5 illustrates the basic representations from the graph, showing player transitions from one state to another. Basic elements from the graph include nodes (a), which represent states, directed edges (b) representing player's actions, meta-edges (c), and player icon (d) representing time-dependent location of individual players. The red meta-edge near the center of the graph is composed of two edges, "mirror" (blue edge) and "rotate" (red edge), as can be seen by the window with the blue and red bars. This window also details the meta-edge composition ratio: 28.25% of the meta-edge's thickness is due to the number of "mirror" edges in it and 71.15% is from the edge "rotate".

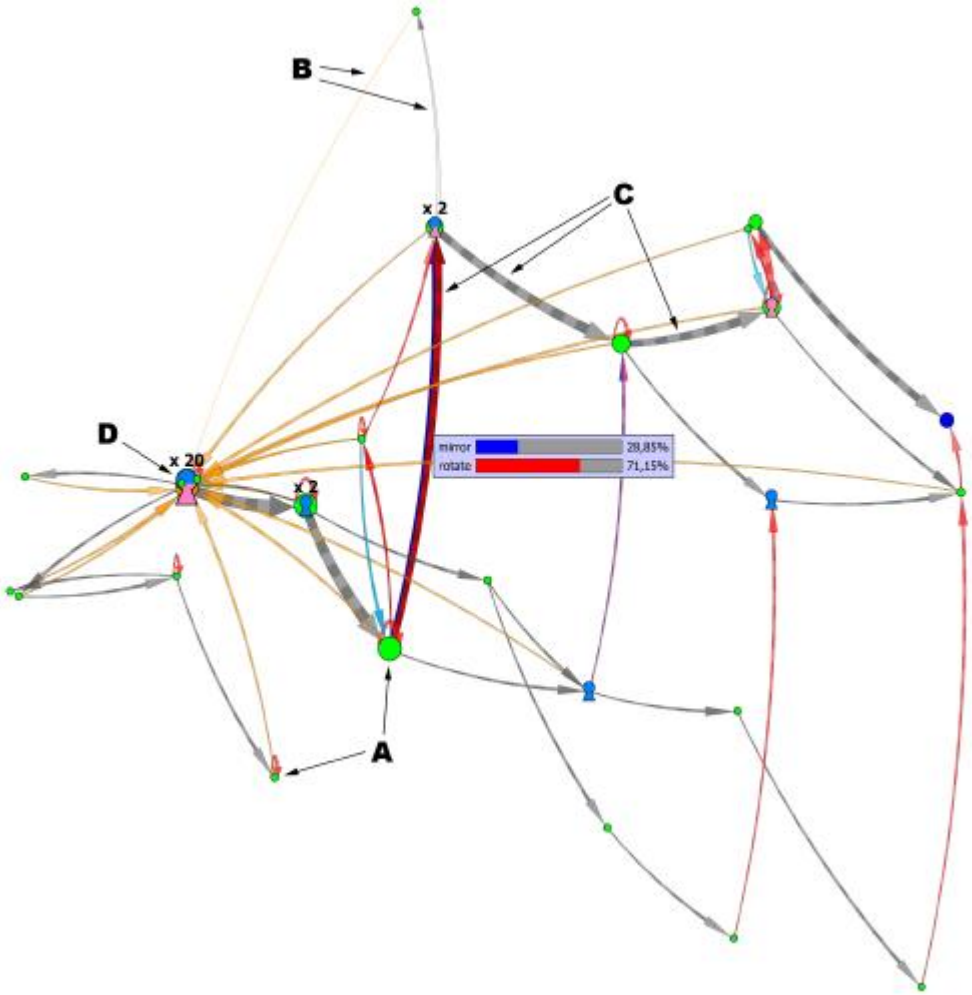

Figure 5: Basic elements from the Play-Graph. Figure taken from WALLNER (2013).

Viewing gameplay data by graphs allows the usage of graph theory concepts to study and understand player behaviors. The displayed graph from Play-Graph illustrates the player's progression in the game, demonstrating actions he made to change states. Unlike Playtracer, this approach uses temporal information to distinguish states, allowing for observations related to time-dependable events. It also allows for comparing two different graphs from the game context, such as two versions of the same game, one before adding new features and another after. This allows for highlighting areas where player activity has increased or decreased with the new additions.

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/she interacted with the world (*i.e.* combat rounds from the battle against the boss). From the available documentation, 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. 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.

2.6 COMPARISON BETWEEN APPROACHES

This section compares the described approaches by outlining their features according to the following characteristics:

- **Graph**: Indicates if the approach explicitly uses graph to represent information.
- **Graphics**: Indicates if the approach explicitly uses graphical charts to represent information.
- **State Machine**: Indicates if the approach displays state transitions.
- **Data Logging**: Indicates if the approach includes the data logging process.
- **Event Context**: Indicates if the approach gathers contextual information from events and actions that can be used to improve understanding of the event/action.
- **Player Behavior**: Indicates if the approach explicitly displays information from multiple players for a visual analysis of player behaviors.
- **Actions**: Indicates if the approach collects details about the actions performed in the game, instead of only collecting the action's outcomes.
- **Statistical data mining**: Indicates if the information gathered and/or displayed by the approach is used for statistical mining (gameplay metrics).
- **Developer-Oriented**: Indicates if the approach can be used for playtesting (validation).
- **Player-Oriented**: Indicates if the approach can be used by players in order to better understand the game flux.
- **Cause-Effect**: Indicates if the approach gathers information about factors that influenced actions, affecting the outcome.

Table 12 provides a comparative chart among the presented approaches. Fields filled with "√" indicates the approach supports the specified characteristic. Fields with "√" indicates the approach generates sufficient information that can be used for the specified characteristic. However they do not explicitly use or treat the information. Fields filled with "?" indicates

that it was not possible to infer whether the feature is present in the approach through the available documentation.

Table 1: Comparative chart among approaches

Features	GVM (JOSLIN *et al.*, 2007)	TRUE (KIM *et al.*, 2008)	Playtracer (ANDERSEN *et al.*, 2010)	Play-Graph (WALLNER, 2013)
Graph			√	√
Graphic	√	√		
State Machine		?	√	√
Data Logging	√	√	?	?
Event Context		√		
Player Behavior	?	√	√	√
Actions		√		√
Statistical Data Mining	√	√	√	√
Developer-Oriented	√	√	√	√
Player-Oriented				
Cause-Effect				

2.7 FINAL CONSIDERATIONS

This chapter presented existing approaches for game flux analysis, also known as game telemetry. The first approach proposed a framework detailing data logging methods to gather information for analysis. The second approach proposed the usage of contextual information during the gathering process to aid in understanding the events in the game. The third approach uses graph visualization to display aggregated user behavior and to aid in understanding common strategies adopted by players. The last approach also uses a graph to describe and visualize gameplay data and, unlike the previous approach, uses temporal information to observe time-dependable information, such as player distribution at different instances.

These approaches are aimed at players' behaviors, game balancing, and game testing by identifying issues. Furthermore, these approaches use graphs or graphics to illustrate the

general population behavior in order to be used by designers to improve the game. Moreover, they do not capture cause and effect information. They only capture the executed action and not the factors that influenced it. Thus they are not meant to be used for a player perspective, allowing players to better understand the consequences of their actions and the influences each action generated in the game.

This motivated us to create a new approach that follow the player's actions more closely and with higher granularity, stating the player's interactions during the game and which actions influenced the outcomes. Unlike TRUE, which gathers contextual information to determine possible influences, our proposed approach captures the cause and effect relations between actions. This allows for understanding and tracking the consequences of each action and how they affected the outcomes. In other words, we would be recording the provenance of the player's story, which allows for a deeper understanding of how the story progressed by using a provenance graph representing the data gathered during the game session.

Provenance is used in relation to works for art, archeology and paleontology to describe an object's life cycle. However, provenance also began to be used for digital information. Provenance information is expressed by the means of a provenance graph, which is a directed graph with vertices representing entities and edges representing causal relations. Unlike the graphs presented in this chapter, the provenance graph uses different types of vertices and edges to discern information, without compromising legibility. 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.

CHAPTER 3 – PROVENANCE

3.1 INTRODUCTION

Results of scientific experiments cannot be understood without the knowledge of the meaning of data and circumstances occurred during their creation. This type of knowledge includes data provenance (DAVIDSON; FREIRE, 2008; FREIRE *et al.*, 2008). Provenance is well understood in the context of art or digital libraries where historical documentation refers to an object's life cycle (PREMIS WORKING GROUP, 2005). However, for digital provenance there are two types of provenance perspectives: retrospective and prospective (FREIRE *et al.*, 2008). The prospective provenance focuses on specifications and the necessary steps to achieve the generated data, while retrospective provenance focuses on the executed steps and external information used to derive the data.

Recently, data provenance in scientific experimentation has become an important topic in scientific research and, as consequence, workshops and conferences for the subject were specifically created (SIMMHAN *et al.*, 2005). The *International Provenance and Annotation Workshop* (IPAW) (MOREAU *et al.*, 2002) was one of the first data provenance workshops to be created. In each edition, the scientific community lists challenges of data provenance to be solved and receives scientific works with possible solutions. During IPAW'06, participants were interested in questions about provenance for the usage in digital data, involving topics related to documentation, data annotation, and data derivations (BOSE *et al.*, 2006). As a result, the first model of digital provenance, the *Open Provenance Model* (OPM) (MOREAU *et al.*, 2007), was created. The OPM has been designed to address the issues raised during the *Provenance Challenge* (MILES *et al.*, 2010), such as understanding provenance systems and how to express provenance information. It is based on retrospective provenance.

Later, another provenance model, PROV (GIL; MILES, 2010), was developed by the provenance incubator group at W3C (GIL *et al.*, 2009), which is derived from OPM. According to the group, provenance of digital objects represents the object's origins and PROV is a proposed specification to represent these provenance records. These records contain descriptions of the entities and activities involved in producing and delivering or otherwise influencing a given object. PROV is also focused on retrospective provenance, similarly to OPM. The usage of provenance, regardless of the model, provides a critical foundation for assessing the authenticity of data, enabling reliability and reproducibility, and is a crucial component of workflow systems (GIL *et al.*, 2007; GROTH; MOREAU, 2010).

When PROV was proposed, the OPM model was already being used in several approaches. However, the fact that PROV is supported by the W3C points to the possibility of becoming the de facto provenance model and the migration from OPM to PROV a possibility in the near future (BIVAR *et al.*, 2013). Nevertheless, the aim of this chapter is to present a study of both digital provenance models, which uses retrospective provenance, as well as comparing these models, pointing out their similarities and differences.

As such, this chapter is organized as follow: Section 3.2 provides a guiding example to aid understanding of provenance relationships in a provenance graph. Section 3.3 and 3.4 describe the OPM and PROV, respectively. Section 3.5 compares both digital provenance models and, lastly, Section 3.6 presents the final considerations of this chapter.

3.2 GUIDING EXAMPLE

The following example is used for illustrating and exemplifying terminologies of provenance in the remaining of this chapter. The provenance graph that represents the following example uses OPM notations.

Cake is often a desired dessert for receiving guests or special occasions and is generally easy to make. There are countless varieties of cakes but they tend to follow the same basic steps: Mixing, baking, and decoration. The first step, mixing, is responsible of mixing the cake's ingredients in order to make cake batter. Basic ingredients used in this stage are usually butter, flour, sugar, and eggs. After the cook mix the ingredients for a period of time, generally until the cake batter acquires a uniform color, the batter goes to a cake pan in order to undergo the next stage: baking.

The baking stage takes the cake to the oven to bake the batter. Generally the baking process takes around 30 to 45 minutes, depending on the oven. After baking is complete, the cake must be left untouched to cool down and to be removed from the cake pan. When it is removed from the pan, the cake is ready for the last stage, which consists on decorating it with icing or any other eligible ingredients. Finally, when the decoration is over, the cake is ready to be served as dessert for house guests.

3.3 OPEN PROVENANCE MODEL

The OPM emerged as a result of the *Provenance Challenges* proposed in the context of IPAW. The *Provenance Challenges* came in four editions, one for each year from 2006 to 2010, except 2008, and OPM mainly resulted from the first two challenges and was used on the third challenge. The 1st challenge was focused on providing a forum for the community to

understand the capabilities of different provenance systems and express their provenance representations. The 2nd challenge was aimed at establishing interoperability between systems through exchange of provenance information. With the OPM development, the 3rd challenge was designed to evaluate the OPM in a practical setting, from an interoperability perspective. The last challenge was cancelled because the 3rd challenge was still unsolved and due to the successful launch of another provenance model (PROV) by the W3C group.

The OPM is a provenance model designed to allow provenance information to be exchanged among systems using a common provenance model. It allows developers to build and share tools for such provenance model, while supports a digital representation of provenance for anything, whether it was digitally generated or not. Lastly, OPM defines a set of rules to identify valid inferences that can be done on provenance representations (MOREAU *et al.*, 2007).

OPM assumes that the provenance of objects is represented by a causality graph, which is a directed acyclic graph with annotations to capture more detailed information about the objects' execution. According to MOREAU *et al.* (2007), a provenance graph is "*a record of a past or current execution, and not a description of something that could happen in the future*". Provenance, according to OPM, is composed of three types of elements (artifacts, processes, and agents) connected by causal relationships. The following sections describe in details those aspects as well as the possibility of inferring provenance statements to simplify the provenance graph, if necessary.

3.3.1 TYPES AND RELATIONS

The causality graph used by the OPM is composed of vertices, which can represent *Artifacts*, *Processes*, and *Agents*, and edges that represent causal relationships between vertices. The definitions for all three vertex types are described in the following: *"**Artifacts** are immutable pieces of state that can represent a physical object or its digital representation in a computer system. **Processes** are actions or a sequence of actions performed or caused by artifacts resulting in new artifacts."* Lastly, *"**Agents** are contextual entities acting as a catalyst of a process that can enable, facilitate, control, or affect its execution"* (MOREAU *et al.*, 2007).

The edges of the provenance graph represent causal relationships between its source and destination. The edges can belong to one of the five categories defined by MOREAU *et al.* (2007) and are described at Figure 6. Using the OPM notation, it is possible to generate a provenance graph for the cake's baking process, which is illustrated at Figure 7. In the cake

example, the ingredients, the oven, and the cake are *artifacts*. The mixing, baking, and decorating stages are *processes*. Lastly, the *agent* is the cooker.

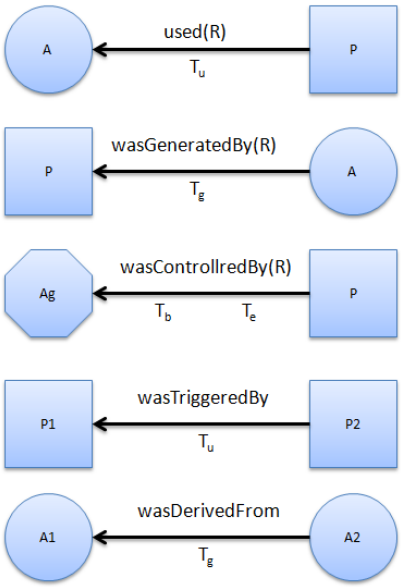

Figure 6: Edges in OPM. Adapted from MOREAU *et al.*(2007).

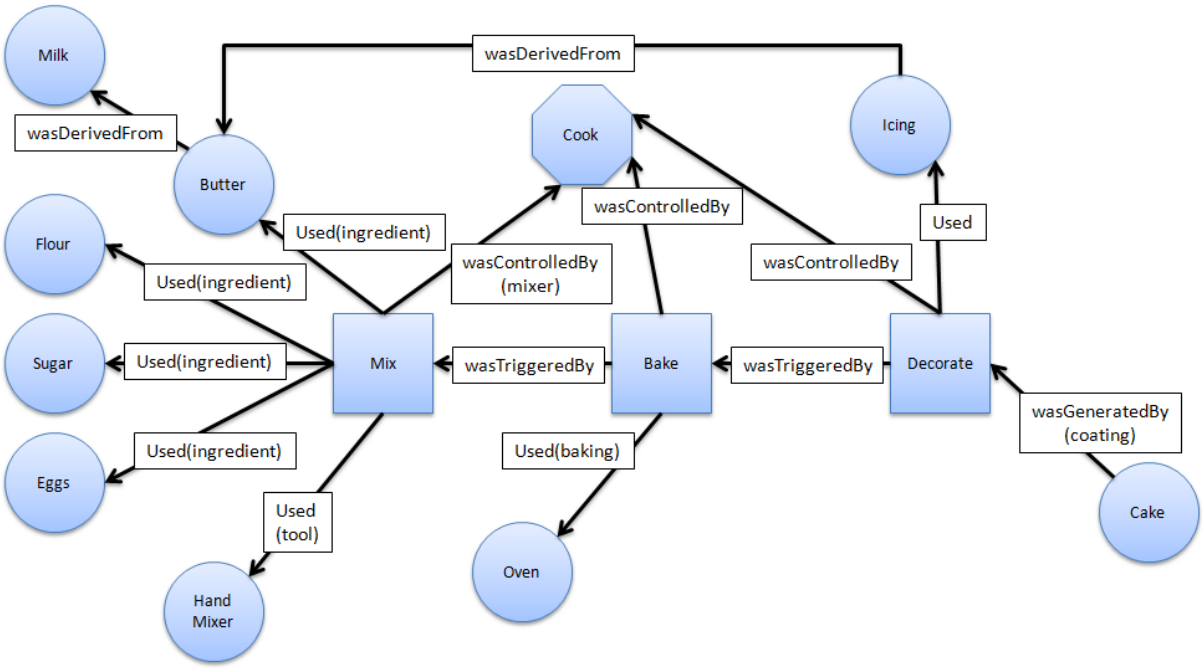

Figure 7: A simplified cake's provenance graph.

A *used* edge that connects a *process* with an *artifact* indicates that the *process* required the *artifact* in order to complete its execution. If several *artifacts* are connected to the same *process*, then all of them were required for the *process* execution. It is also possible to specify each *artifact* role in the *process* execution by using the *(R)* field in the edge, shown in Figure 6, which represents a role. Roles are not defined in OPM but are instead defined by the

application domain. Using the cake example, the mix *process* used butter, flour, sugar, eggs, and a hand mixer. Their roles in the mix *process* are ingredient (butter, flour, sugar, and eggs) and tool (hand mixer).

The edge *wasGeneratedBy* connecting an *artifact* with a *process* indicates that the *artifact* was only generated after the *process* started its execution. If several *processes* are connected with an *artifact*, then all these *process* must have started for the *artifact* to be generated. Similar to *used* edge, it is possible to specify the *process's* role in the generation of the *artifact*. In the cake example, the cake was generated by the *process* decorate with the role of coating the cake with icing to be a more appealing dessert for guests, instead of a simple cake.

A *wasControlledBy* edge connecting a *process* to an *agent* indicates that the *agent* controlled the *process's* start and ending. If the *process* was controlled by multiple *agents*, then their roles can also be represented in the relationship. Note however that this type of dependency represents a control relationship and not a data derivation. In the cake example, the cook control the executions of the stages mix, bake, and decorate. He decides how long is the mixing, how long it takes to bake the cake, and how to decorate it.

The edge *wasTriggeredBy* connecting a *process* with another *process* indicates that the *process* was only able to complete after the other *process* started. This edge also allows for a *process* to have an oriented view of past executions. Using the cake example, the *process* bake was only able to start after the *process* mix had started and, in this case, finished as well.

Lastly, the edge *wasDerivedFrom* connecting an *artifact* with another *artifact* indicates that the first *artifact* had to be generated for the second *artifact* to be generated. Using the cake example, the butter is made by churning milk. In other worlds, milk is necessary for the creation of butter, this way, butter was derived from the milk.

Moreover, OPM allows causality graphs to be used with time information. In OPM, time can be used to validate causality claims because if the same time clock is used to measure the passage of time for the effect and cause, then the effect will always come after the cause. In addition, time can be associated to *instantaneous occurrences* in a *process*. There are four types of occurrences (MOREAU *et al.*, 2007): *creation*, *use*, *starting* and *ending*. The first two are used for *artifacts* while the other two for *processes*. Because time is measured by someone, the model allows a margin of error when representing time. For example, an *artifact* was used no earlier than time t_1 and no later than time t_2. This rationale is analogous for *processes*.

Figure 6 indicates that time information can be expressed in the model by using labels in the relationships. For *used* and *was generated by* edges, one timestamp can be used to express when the event happened. For *was controlled by* edge two timestamps mark when the process started and terminated. For *was derived from* and *was triggered by* edges, one timestamp to indicate when the *artifact* was generated. Despite using timestamp, the time of occurrence itself is not enough to imply causality. The fact that *process* P_1 happened before P_2 is not enough information to infer that P_1 is the cause for P_2 execution.

3.3.2 INFERENCE

The OPM also has defined the notion of a graph based on a set of syntactic rules and topological constraints (MOREAU *et al.*, 2007). The provenance graph captures causal dependencies that can be summarized by inferring facts, using transitive rules. Because of this, the OPM defined a set of completion rules and inferences that can be used in the graph to improve understanding by omitting unnecessary steps. There are three completion rules: *artifact introduction*, *artifact elimination*, *process introduction*, and *process elimination*.

For *artifact introduction*, Figure 8 illustrates the transformation of the *wasTriggeredBy* edge to *wasGeneratedBy* and *used* edges, introducing an *artifact* (cake batter). This is possible because the *wasTriggeredBy* edge is a composition of both edges. However, the identity of the *artifact* cannot be specified due to the lack of information, unless if the same *artifact* was previously present in the graph and removed by inference or specified somewhere else. Analogous, in the *artifact elimination* it is possible to transform the *used* and *wasGeneratedBy* edges to *wasTriggeredBy* edge by hiding the *artifact* (cake butter). This is possible because *wasTriggeredBy* edge is a composition of both edges.

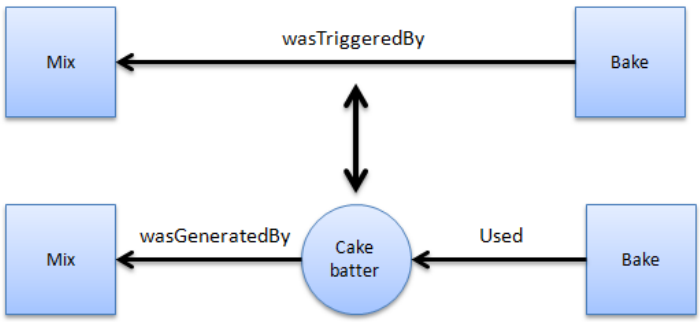

Figure 8: Artifact introduction and elimination using the cake example.

Another rule is the *process introduction* illustrated by Figure 9, which is similar to *artifact introduction* but with a *process* instead of an *artifact*. While the *wasTriggeredBy* edge hides the presence of an *artifact*, the *wasDerivedFrom* edge hides the presence of an

intermediary *process*. However, the *process introduction* rule can only be applied if butter is actually dependent of milk. This is the case, since in order to produce butter, it is necessary to churn the milk. So it is possible to add the *churning process* between milk and butter. The last rule, *process elimination*, is analogous to *artifact introducion*.

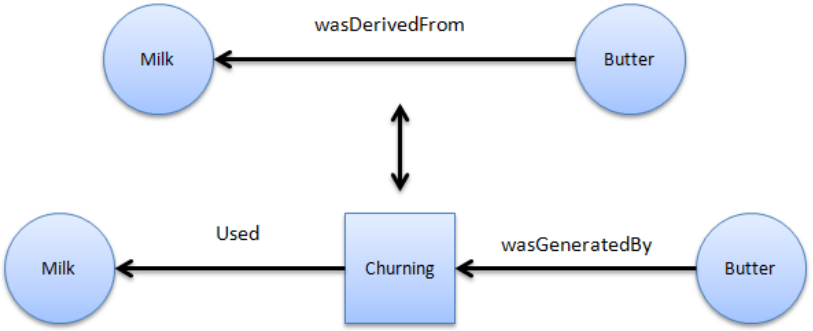

Figure 9: Process introduction and elimination in the cake example.

In a provenance graph, the causes of an *artifact* or a *process* might be traced to indirect causes from other *processes* and can involve multiple transitions between the cause and the perceived effect. For this purpose, OPM offers a set of relationships necessary for inferences: the *Multi-step edges* (MOREAU *et al.*, 2007). The multi-step edges can be composed of four types: *wasDerivedFrom**, *used**, *wasGeneratedBy**, and *wasTriggeredBy**. Unlike their normal counterparts, these edges represent that multiple steps were necessary to be taken in order to have the same meaning as their normal counterparts.

Using the cake example, it is possible to make inferences by using multi-step edges, which are represented as dashed edges in Figure 10. The edge *wasDerivedFrom** says that the *artifact* icing was indirectly derived from milk (using a multi-step edge), because the butter, which the icing is derived from, is in turn derived from milk. This states that the *artifact* milk had an influence over the *artifact* icing. Another possible multi-step is the *wasGeneratedBy** connecting *artifact* cake to *process* mix. This is possible because the *artifact* cake was generated by *processes* decorate, bake, and mix. Moreover, *process* decorate *wasTriggeredBy* process* mix, since *process* decorated could only have started after *process* mix had started, and ended, in this example. Lastly, *process* bake *used* artifact* eggs because eggs was necessary for the *process* mix, which in turn was necessary for *process* bake to start. As can be seen from these examples, multi-step edges can be used to infer single-step edges by eliminating *artifacts* and *processes* that occur in a chain of events.

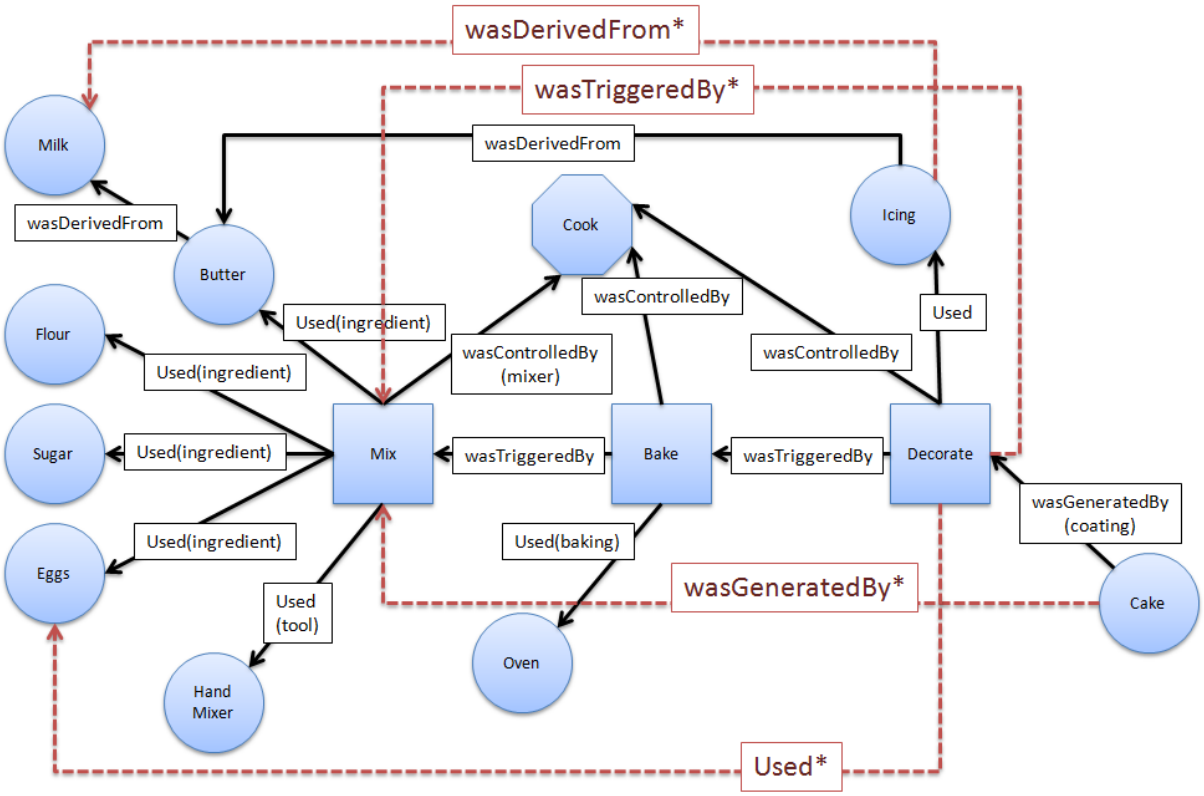

Figure 10: Multi-step edges in the cake example.

Lastly, the OPM has a modular design as illustrated by Figure 11. However, specifications for all layers in the design have not been produced, possibly because the development team started working on another provenance model: PROV. Nevertheless, at the bottom layer is located the abstract model (MOREAU *et al.*, 2007). On the left hand side, is located a serialization to *xml*, defined by OPMX (*The Open Provenance Model XML Schema)* (MOREAU; GROTH; *et al.*, 2010), a mapping to RDF with OPMV (*The Open Provenance Model Vocabulary)* (ZHAO, 2010) and OPMO (*The Open Provenance Model OWL Ontology)* (MOREAU; DING; *et al.*, 2010). Those are the only produced specifications, along with the *Open Provenance Model Java Library* (MOREAU, 2010b), and a JAXB-generated Java Library used by *OPM Toolbox* (MOREAU, 2010a) for creating a Java representation of

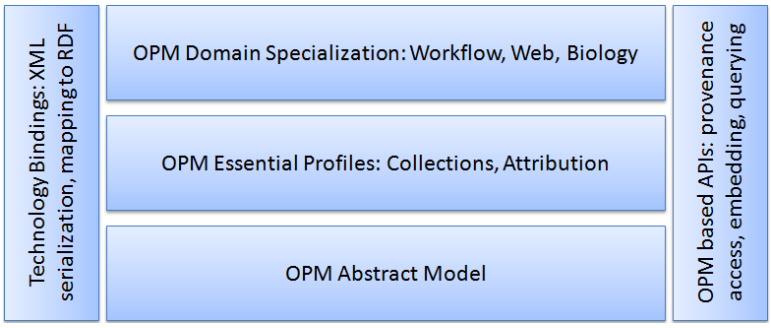

Figure 11: OPM's Layered Architecture. Adapted from MOREAU *et al* (2007)

OPM graphs and serializing them to or from a *XML* file. With the development of PROV, these other OPM specifications (Essential Profiles, Domain Specialization, and APIs) were left unfinished.

3.4 PROV

PROV is a family of specifications proposed by W3C group to express provenance of digital objects, containing descriptions of the entities and activities involved in producing and delivering an object. In PROV, "*provenance is information about entities, activities, and people involved in producing a piece of data which can be used to form assessments about its quality, reliability or trustworthiness*" (NIES *et al.*, 2010). The goal of this provenance model is similar to OPM, allowing provenance information to be exchanged among systems using a common provenance model, while also enabling the provenance representation by a provenance graph (GROTH; MOREAU, 2010).

The discussion group that developed PROV was officially launched concurrently to the forth *Provenance Challenge* (MILES *et al.*, 2010). The specifications that make up the PROV provenance model were divided into several documents that details different aspects of it: PROV *Overview (PROV-OVERVIEW)* (GROTH; MOREAU, 2010), PROV *Primer (PROV-PRIMER)* (GIL; MILES, 2010), Prov *Ontology (PROV-O)* (LEBO *et al.*, 2010), PROV *Data Model (PROV-DM)* (MOREAU; MISSIER, 2010a), PROV *Constraints (PROV-CONSTRAINTS)* (NIES *et al.*, 2010), PROV *Notation (PROV-N)* (MOREAU; MISSIER, 2010b), PROV XML (PROV-XML) (HUA *et al.*, 2010), PROV *Dublin Core Mapping (PROV-DC)* (GARIJO *et al.*, 2010), PROV *Links* (MOREAU; LEBO, 2010), PROV *Access and Query (PROV-AQ)* (WEITZNER *et al.*, 2008), PROV *Dictionary (PROV-DICTIONARY)* (MISSIER *et al.*, 2010), PROV *Semantics (PROV-SEM)* (CHENEY, 2010), and PROV *Implementations* (GROTH *et al.*, 2012) .

Figure 12 illustrates the organization of PROV. At its core, is the conceptual data model that defines the common vocabulary used to describe provenance. Inside the data model, there is a set of constraints defined to aid developers in creating provenance programs to validate provenance statements. In order to support the interchange of provenance, PROV defined protocols to locate, access, and connect sets of provenance in order to aid in their interoperability.

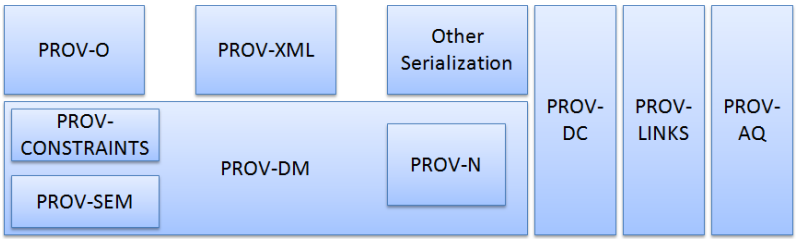

Figure 12: PROV organization. Adapted from GROTH and MOREAU (2010)

Provenance can be used for many purposes, from understanding how the data was collected in order to use it, to determine the object's ownership, and decide if the information is trustworthy. It can also be used to check if the steps used in the process to obtain the result is compatible with the requirements. Mainly, it is used to show the necessary steps to reproduce something. The PROV model specification accommodates these usages of provenance and provides three different ways to capture information according to the user's perspective of provenance (GIL; MILES, 2010): agent-centered, object-centered, and process-centered.

The agent-centered approach focus on describing entities involved in the generation or manipulation of the information, while process-centered focus on capturing actions and steps used to generate the information. Lastly, the object-centered approach traces the origins of a document or artifact to other artifacts.

3.4.1 TYPES AND RELATIONS

PROV also uses a graph, similar to the provenance graph from OPM, to represent the provenance information. This graph is also characterized by having edges representing relationships between vertices and three types of vertices (GIL; MILES, 2010): *Entities*, *Activities*, and *Agents*.

Similarly to *artifacts* from OPM, *entities* represent physical or digital objects like a document, the web, or material objects. *Activities*, which are similar to *processes* in OPM, are actions taken to change or interact with *entities* or *agents*. Lastly, an *agent* is a person, software, organization, or *entities* that have responsibilities. This responsibility is a link to an *activity*. Several *agents* can have responsibilities over the same *activity* and a single *agent* can have responsibilities over several *activities*. *Agents* can also act on behalf of other *agents*, representing their interests when they are unavailable. These causal relations are some of the possible relationships available in PROV and are represented by edges in the provenance graph, similarly to OPM. Figure 13 illustrates those vertex types in the graph along with some

possible relations between them. These relations, as well as other possible relations, are defined in the following paragraphs according to PROV-DM (MOREAU; MISSIER, 2010a).

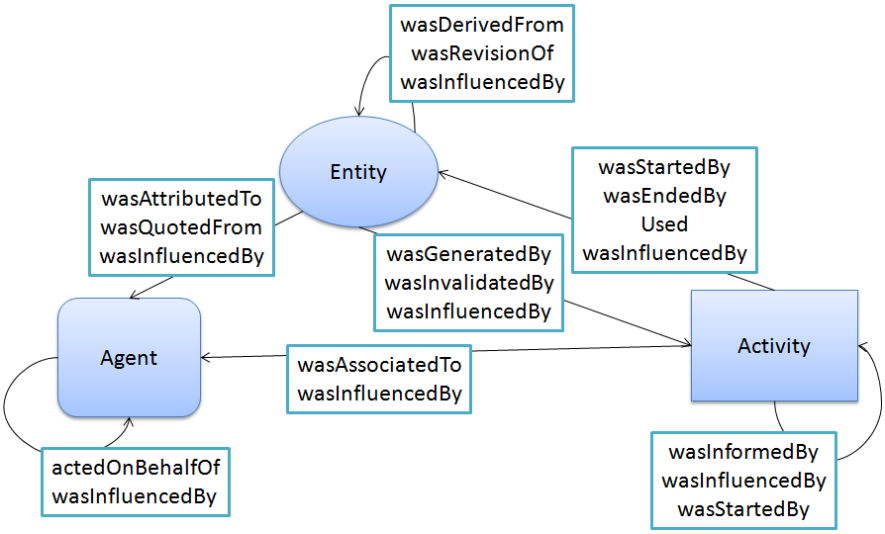

Figure 13: PROV Entities and possible relations. Adapted from GIL and MILES (2010)

Just like in OPM, the relationship denoted by a *used* edge from an *activity* to an *entity* indicates that the *activity* needed the *entity* for its operation. The relationship denoted by *wasStartedBy* edge from an *activity* to an *entity*, or between *activities*, indicates that the *entity* was the trigger that started the *activity*, similar to *wasTriggeredBy* from OPM. Similar to *wasStartedBy*, the *wasEndedBy* relationship indicates when the *activity* ended its execution by an *entity* or another *activity*.

The relationships *wasGeneratedBy* and *wasDerivredFrom* are also similar to their OPM's counterparts. The *wasInvalidadedBy* relationship between *entities* indicates that the *entity* is no longer available for usage. For example, in the cake scenario if the butter was home-made and there was only enough milk to make it for the cake and not the icing, then the milk was invalidated by the butter, since there is no more milk left for making the icing. The *wasInformedBy* edge indicates an exchange of information using an unknown *entity* between *activities*. Using the cake example but in a scenario where the process is fully automated, this edge could be used for another *activity* to inform the bake *activity* how long the cake batter should bake to not ruin the cake. The relationship *wasAttributedTo* from an *entity* to an *agent* indicates that the *entity* is now assigned to that *agent*, while *wasAssociatedTo* from an *activity* to an *agent* indicates that the *agent* had a responsibility or a role in the *activity*.

The relationship between *agents* denoted by *actedOnBehalfOf* indicates that an *agent* assigned authority to another *agent* to execute *activities* on his behalf. The edge *wasRevisionOf* between *entities* indicates that the *entity* is a previous version of another *entity*. The edge *wasQuotedFrom* indicates that an *entity* was used by an *agent* and that agent was

not its original owner. The *entity* must be a text or image taken from somewhere else, just like quoting what someone wrote. Lastly, the *wasInfluencedBy* edge indicates that the *entity*, *activity*, or *agent* affected another *entity*, *activity*, or *agent* by influencing it.

3.4.2 FURTHER NOTATIONS

Besides the relations mentioned in the previous subsection, the PROV model has support for a few more features: specialization, alternate, and the possibility of extending existing structures. These extended structures are defined by a variety of mechanisms: subtyping, expanded relations, new relations, and optional identifiers.

According to MOREAU and MISSIER (2010a), an specialization, denoted by a *specializationOf* edge from an *entity* to another *entity* indicates that the first *entity* can do everything the second *entity* can, while also having more functions. Meanwhile, the *alternateOf* edge between *entities* indicates that both *entities* have the same functions and characteristics. Specialization and alternate are not considered as influences in PROV. Subtyping is a rule to create new edges based in existing relationships, *entities*, or *agents*. For example, the revision relationship is also a subtyping of the derivation relationship since a revision is a newer version based on the original document. PROV also supports the creation of totally new relationships without using any of the existing ones as a basis, which may be useful depending on the domain. For example, the relation *wasAttackedBy* is a relation between *agents* that indicates that one *agent* suffered an attack action from another *agent* in the game domain.

The relationships represented by edges between two vertices can also be expanded to add more details. For example, in a revision relationship, it is possible to add further details about the changes between versions, describing how the entity was altered to generate the newer version. Another possible example is planning, used by agents. A planning, which in PROV can be represented as an entity, is a set of necessary steps to be taken in order to achieve the proposed goal. Figure 14 illustrates the usage of the *expanded relation plan* in the *hadPlan* edge connecting the edge *wasControlledBy*, from an *agent* (Cook) and an *activity* (Mix), with an *entity* (Instructions: Recipe).

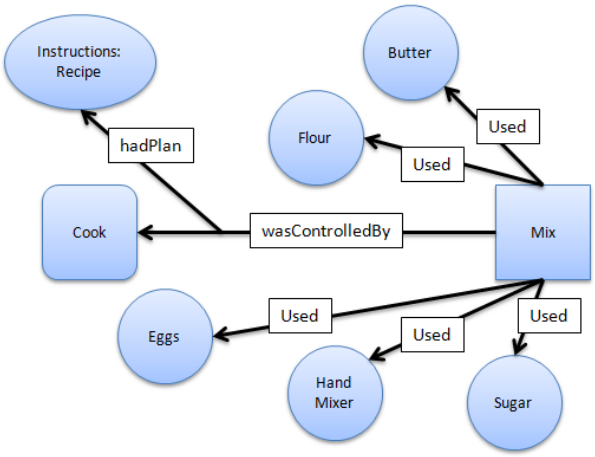

Figure 14: Using *Expanded Relations* in the cake example.

The PROV data model also has a set of pre-defined attributes to be used as optional identifiers to provide further details. There are five different types of attributes: *label*, *location*, *role*, *type*, and *value*. Labels are used to name an *agent*, *entity*, or *activity* for easier understanding, while locations are used to identify places that the agent, entity, or activity was located. Role, as previously mentioned, can be assigned to an *entity* or *agent* to provide an insight about their responsibilities during the execution of an *activity*. Type provides further information about the *agent*, *entity*, or *activity*. For example, *software agent* is a type of *agent* and chocolate cake is a type of cake. Finally, value is used to represent values associated with the entity, which can be a string, a number, or encoded data. Table 2 describes which constructions are allowed for each attribute and if there is any restriction of their values.

Table 2: PROV optional attributes. Adapted from MOREAU and MISSIER (2010a)

Attribute	Allowed in	Allowed Values
label	Any construction	*Value* of type *String*
location	*Entity*, *Activity*, *Agent*, *Usage*, *Generation*, *Invalidation*, *Start*, and *End*	*Any value*
role	*Usage*, *Generation*, *Invalidation*, *Association*, *Start* and *End*	*Any value*
type	Any construct	*Any value*
value	*Entity*	*Any value*

3.4.3 TIME INFORMATION

The PROV model offers the ability to store information data from the time of origin due to the importance of temporal information in some scenarios. It is allowed to store date and time relating to *entities* or *activities*. For *entities*, it is allowed to store information regarding its generation or usage. As for *activities*, it is allowed to store information regarding when it started and ended its execution.

This information can be stored in tickets in the *activity* or in the relationships, as illustrated by Figure 15, showing the *startedAtTime* and *endedAtTime* in the *activity* bake, and the *generatedAtTime* in the relationship *wasGeneratedBy*. These tickets can also be used to store other information details, such as the usage of *Optional Identification* as mentioned in the previous subsection.

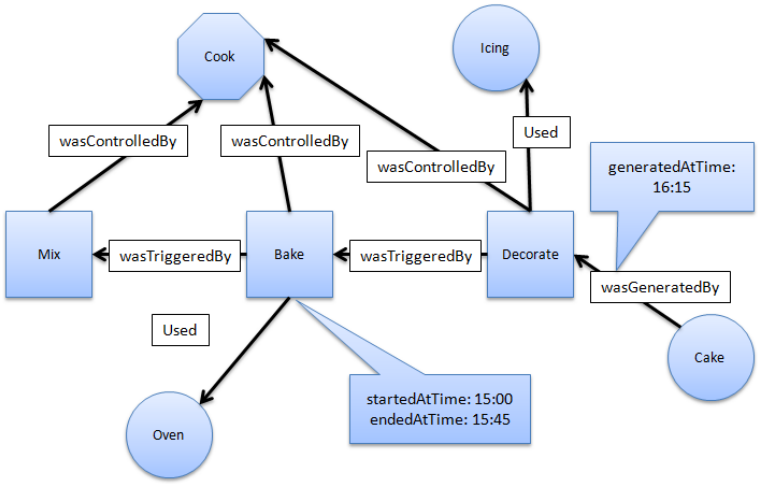

Figure 15: Time information using the cake example.

3.4.4 INFERENCE

Like OPM, the PROV model also supports the usage of inferences on provenance data to identify indirect effects or influences, while also preserving the meaning. A set of rules to validate a provenance inference in PROV is available at NEIS et al. (2010). An inference in PROV is a rule to add new statements or edges in order to simplify the information by skipping statements. It is also possible to replace statements or edges for equivalent ones to improve readability without changing the meaning of the statement (NIES *et al.*, 2010).

For example, it is possible to change a *wasDerivedFrom* relationship to a *wasRevisionOf* relationship if the meaning of the derivation in that situation is equal to a revision. This type of equivalence in PROV is called as *definitions*. Both definitions and inferences can be viewed as logical formulas (NIES *et al.*, 2010). However, while a *definition* can be used for both directions of an implication, this is not true for inferences. While looking at the *activities* necessary to generate a cake (*mix*, *bake*, *decoration*), it is possible to infer that the cake was generated by the *activity* mix. However, from the perspective of the *activity mix*, it is not possible to infer that *activities bake* and *decoration* were also involved.

3.5 COMPARISON BETWEEN MODELS

In terms of key elements from both provenance models, it is possible to make a direct mapping between concepts by associating *artifacts*, *process*, and *agents* from OPM to *entities*, *activities*, and *agents* of PROV, respectively. Both models present ways of marking the passage of time (with timestamps) and execution, as well as providing rules for making inferences. However, PROV also provide explicit support for extending existing features, such as *subtyping*, expanding, and creating new relationships. Some relationships from both models are also compatible because, aside from having the same names, they also carry the same causal relationship between objects. These common relationships are: *used*, *wasGeneratedBy*, and *wasDerivedFrom*.

However, the relationship *wasControlledBy* from OPM doesn't have one from PROV with the same name, but the relationship *wasAssociatedWith* from PROV has the same function, linking *activities* (*processes* in OPM) to *agents* in both models. Despite both relationships *wasTriggeredBy* from OPM and *wasInformedBy* from PROV being between two *processes* or *activities*, in PROV, they have different purposes. The relationship *wasInformedBy* aims to show that a particular *activity* reported something to the other, while *wasTriggedBy* from OPM indicates that a *process* has been initiated by another. However, there is a relationship in PROV (*wasStartedBy*) which equals to *wasTriggedBy* from OPM. Although the relationship *wasStartedBy* is more comprehensive as it can occur not only between two *activities*, but also between an *entity* and an *activity*.

Also, the PROV model has four relationships that were not found in OPM: the aforementioned *wasInformedBy*, *wasEndedBy*, *actedOnBehalfOf*, and *wasAttributedTo*. The relationships *actedOnBehalfOf* and *wasAttributedTo* are delegation and association of *agents* to *entities* and *activities*. These are important because PROV also aims at providing provenance information centered on *agents*, something that does not occur naturally in OPM. Lastly, the relationship *wasEndedBy* aims to represent the *activity's* finalization. Table 3 illustrates the comparison of the existing relationships from both provenance models.

Table 3: OPM x PROV

OPM	PROV
used	***used***
wasGeneratedBy	***wasGeneratedBy***
wasControlledBy	***wasAssociatedWith***
wasDerivedFrom	***wasDerivedFrom***
wasTriggeredBy	***wasStartedBy***
	wasEndedBy
	wasRevisionOf
	wasAttributtedTo
	wasInformedBy
	actedOnBehalfOf

From these relationships without direct equivalences, it is possible to observe differences between models. The OPM is simpler, and apparently aimed at controlling flows of execution taking particular attention on how a *process* was started by another. On the other hand, PROV appears to be more focused on issues of responsibility and historical data, having several relationships between *agents* and the other types (*entities* and *activities*), but also being more complete, having all relationships equivalent to the OPM. This may be due to the fact that the majority of OPM's designers also participated in the creation of PROV.

Aside from these differences in naming and existing relations, both models have core similarities. They distinguish entities in three categories: artifact/entity, process/activity, and agent. However, the major difference is that OPM stopped writing specifications for its features possibly because the PROV model emerged, which core OPM researches were involved in its conception. Even though they are treated as different provenance models, I see them as different stages of development, where OPM is the equivalent of an alpha build and PROV the release of the provenance model.

3.6 FINAL CONSIDERATIONS

This chapter presented the concepts of provenance in order to gather historical information about objects for further analysis. It was also presented both the existing provenance models (OPM and PROV) that can be used for provenance of digital information. Later, a comparison between models was made, pointing out their similarities. The incomplete OPM specifications may be a side effect from the fact that the same designers were involved in the creation of PROV, which occurred around the same year that OPM updates halted (at 2010). By analyzing both provenance models, there are three key points that inspired in the method used in this work for collecting gameplay data and representing the game flux in a provenance graph:

- Provenance of objects, which allows for a detailed study of an object's life cycle.
- Provenance inferences, which allows making statements while, at the same time, hiding unimportant facts to reach conclusions about the object's history.
- Provenance Graph, which allows for an analysis of the object's interactions and influences from other *entities* throughout its life cycle by the means of a graph.

Thus, the next chapter presents the proposed approach to improve understanding of the game flux, providing insights in the story progression and how the outcome was influenced. The proposed approach is based on analyzing the game flux using a provenance graph and is called *Provenance in Games*.

CHAPTER 4 – PROVENANCE IN GAMES

4.1 INTRODUCTION

The conclusion of a game session derives from a series of decisions and actions made throughout the game. In many situations, analyzing and understanding the events, mistakes, and flows of a concrete gameplay experience may be useful for understanding the achieved results. A game flux analysis might be fundamental for detecting symptoms of problems that occurred due to wrong decision-making or even bad gameplay or game balancing design. Without this type of analysis, the player would be required to play the game again and make different decisions to intuitively guess which ones were not adequate to the situation. However, depending on the game dynamics and complexity, reproducing the same state can be unviable, making it difficult to replay and try new solutions.

Additionally, neural studies about the learning capability of human brain (CHIALVO; BAK, 1999; CLARK, 1950) state that the process of learning by correcting past mistakes is efficient and, consequently, desirable for the learning process. This process increases the human ability to adapt to new situations due to the rule of changing synaptic strengths, which ensures that synaptic changes occur only at neurons involved in wrong outputs. Nevertheless, in order to correct mistakes, it is fundamental to know which mistakes occurred.

As previously presented at Chapter 2, traditional games are limited in terms of analysis from the obtained results and, as such, might compromise the player's ability to figure out the effects caused by the actions taken during a game session. Watching the game unfold again, through a replay feature, or looking at statistical graphs might not be enough to understand the reasons that affected the outcome, or how something happened the way it did and not the way it was expected to. For example, why did the player lost his vastly superior army to the enemy's inferior forces? Was it due to the terrain disadvantages? Or was it because of a previously casted spell on the enemy's armies that tipped in his favor? Such questions are common to arise and sometimes their influences are not apparent to the player. Nevertheless, even if they were identified, analyzing them in more details might provide useful insights for future occasions.

The goal of this work is to improve the understanding of the game flux, providing insights on how the story progressed and the influences in the outcome. In order to improve understanding, this work provides the means for analyzing the game flux by using provenance. The provenance analysis is done by processing collected gameplay data and

generating a provenance graph, which relates the actions and events that occurred during the game session. This provenance graph might allow the designer, the player, or a third person, such as a mentor, to identify critical actions that influenced the game outcome and helps to understand how events were generated and which decisions influenced them. This process may also aid in the identification of mistakes, allowing the player to reflect upon them for future interactions.

Thus, this work proposes a conceptual framework that collects information during game sessions and maps it to provenance terms, using digital provenance (FREIRE *et al.*, 2008) concepts for representing the game flux, and providing the means for a post-game analysis. This chapter explains our novel proposal of the *Provenance in Games* (KOHWALTER *et al.*, 2012) conceptual framework, which is divided in two stages: provenance gathering, which gathers gameplay information and generate a *game flux log*[6], and provenance visualization, which uses the *game flux log* to generate a provenance graph.

This chapter is organized as follow: Section 4.2 explains the provenance gathering stage and the storage structured used to generate the *game flux log*, while Section 4.3 explains the provenance visualization stage. Lastly, Section 4.4 presents the final considerations of this chapter.

4.2 PROVENANCE GATHERING

A typical game architecture is mainly composed of game objects and the game loop. All objects present in a game, from environment objects to characters, are inherently game objects. Game objects by themselves do not add characteristics to the game. Instead, they are containers that hold components that implement actual functionality, such as scripts (*i.e.* artificial intelligence, player controller), meshes (the object structure or "body"), physics, textures, animations, and audio. Meanwhile, the game loop is responsible for the sequence of events that occur in a game, allowing the game to keep running regardless of the user's input. The game loop keeps the game alive, updating game object states and executing their actions. Each script in a game object has a function called Update, which is used by the game loop to execute their functions. Every time the game loop is ticked, it executes the Update function from all scripts that belongs to the game objects present in the scene.

In this work, we propose a novel usage for provenance, in the game field. In order to adopt provenance for the context of games, it is necessary to map each type of vertices of the provenance graph into elements that typical can be found in games. As mentioned in chapter

[6] The *game flux log* can also be viewed as the player's journey.

3, the OPM and PROV use three types of vertex: *Artifacts*/*Entities*, *Process*/*Activities*, and *Agents*. In order to use these vertex types, it is first necessary to define their counterparts in the game context. To avoid misunderstanding, we adopt throughout this chapter the terms used in PROV (entities, activities, and agents).

In the context of provenance, *entities* are defined as physical or digital objects. Trivially, in our approach they are mapped into objects present in the game, such as weapons and potions. In provenance, an *agent* corresponds to a person, an organization, or anything with responsibilities. In the game context, agents are mapped into characters present in the game, such as non-playable characters (NPCs), monsters (enemy type of NPCs), and players. It can also be used to map event controllers, plot triggers, or the game's artificial intelligence overseer that manages the plot. Thus, *agents* represent dynamic game objects, while *entities* represent static game objects. Lastly, *activities* are defined as actions taken by agents or interactions with other agents or entities. In the game context, *activities* are defined as actions or events executed throughout the game, such as attacking, dodging, and jumping.

With all three types of vertex mapped into the game context, it is also necessary to map their causal relations to create the provenance graph. The PROV model defines some causal relations that can be used similarly to their original context. However, it also provides rules to extend these relationships or to create new ones. For instance, it is possible to create relationships to express the damage done to a character or relationships that affect specific core mechanics from the game, like attack rolls, healing, and interactions with NPCs or objects. Also, the PROV model deals well with the aspect of time, which can be heavily explored in games, especially on games focused on storytelling.

Each NPC in the game should explicitly model its behavior in order to generate and control its actions, providing an array of behavior possibilities. For example, decision trees (MORET, 1982) can be used to model the NPC's behaviors. With this explicit model, a behavior controller can register information about the action when it is executed. Actions can be represented by a series of attributes that provide a description and the context of the action, allowing the creation of a provenance graph. As illustrated by Figure 16, every action needs some information: a reason for its existence, why the action was performed, what triggered it, and who performed the action. In addition, the time of its occurrence can be important depending of the reason of using provenance. The main reason of using provenance is to produce a graph containing details that can be tracked to determine why something occurred the way it did. Therefore, with this assumption, the time of the action, the person who did it, and the effects of the action can be recorded for future analysis. With few adaptations on the

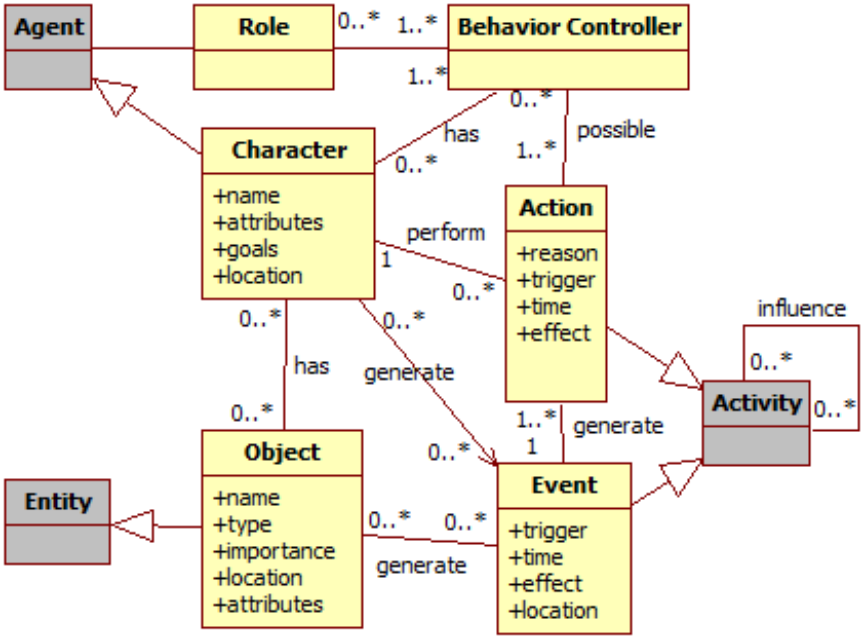

Figure 16: Data model diagram. Gray classes represent generic provenance classes.

information gathered, such as recording velocity, direction, and mass, it is possible to register physics aspects from the game.

For example, a monster attacked the player and scored a hit causing some damage, which in turns decreases the player's hit points (HP). The relevant information for this action is: when it was executed (time, turn, or combat round), who executed it (in this case, the monster), why it was executed (was it a special attack used because his HP was low? Or a normal attack?), who this action affected (in this case, the player), and the consequences of this action (decreased the player's HP). If the action affects more than one character, then it is important to record all people involved and how the action affected each one. For example, suppose that the attack action was actually a buffing attack, which provides a boost to the monster's allies and does damage to the target. In this case, aside from recording the inflicted damage, it should also be recorded the buff received by the monster's allies.

Events also work in a similar way as actions, with the difference in who triggered them, since events are not necessary tied to characters. For objects, its name, type, location, importance, and the events that are generated by it can also be stored to aid in the construction of the graph. Lastly, agents can have their names, attributes, goals, and current location recorded. The information collected during the game is used for the generation of the *game flux log*, which in turn is used for generating the provenance graph. In other words, the information collected throughout the game session is the information displayed by the provenance graph for analysis. Thus, all relevant data should be registered, preferentially at

fine grain. The way of measuring relevance varies from game to game, but ideally it is any information that can be used to aid the analysis process.

4.2.1 STORAGE STRUCTURE

Our conceptual framework requires a data infrastructure for all the necessary data to be used later for the provenance visualization stage. The storage structure is similar to lists. For example, each *agent* has a list of actions that contains all his executed actions. This list structure allows for inferring the *agent* that executed each action by simply looking at whose list the action belongs to, without the need of explicitly saying who was responsible for the execution of the action. Furthermore, actions that were influenced by other actions are connected to each action that influenced it. For events, it is used an analogous approach, storing all events by trigger.

Figure 17 illustrates an example of this structure. By looking at the actions "Cast Spell" in the figure, it is possible to infer that they were executed by the mage because of the black line that links the chain of actions ("Cast Spell: *Weakness*", "Cast Spell: *Stonefist*", "Cast Spell: *Heal*") to the mage. It is also possible to infer the order each action was executed by looking at the structure, since new actions are added at the end of the list. Thus, in the example, the order of spells the mage casted is: *Weakness*, *Stonefist*, and lastly, *Heal*. Similarly, the Orc executed the actions "Charge to Mage", "Attack (Missed)", and "Attack" in that order.

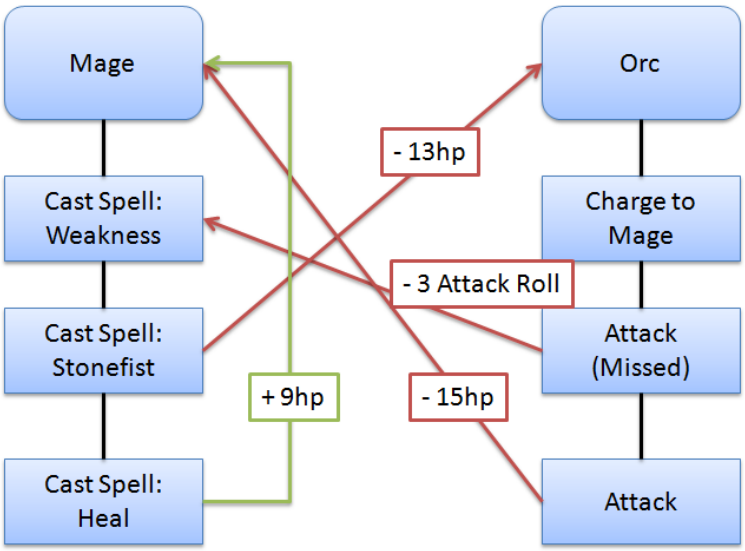

Figure 17: Provenance representation of a combat

If an action generates influence that affect another action then the influenced action will have a connection to the action that influenced it. If an ally used a buff spell on the player

that buffs his attack rate, then, when the player's attack action is generated, it will store the (current) action's details and have a connection to the ally's action that provided the buff. There is no need to explicitly mention the ally because each action belongs to a list, which in turn belongs to an *agent*. This structure allows for inferring who influenced the action by following the connections of each action. If there were multiple influences in the executed action, then it is necessary to store a pointer for each action that influenced it. In the case of an action generating influence, it is necessary to temporarily store a pointer to this action for future actions that might be affected by it.

For example, suppose the battle between a mage and an orc fighter illustrated in Figure 17 and detailed step by step in Figure 18. At the beginning of the battle, the mage casts

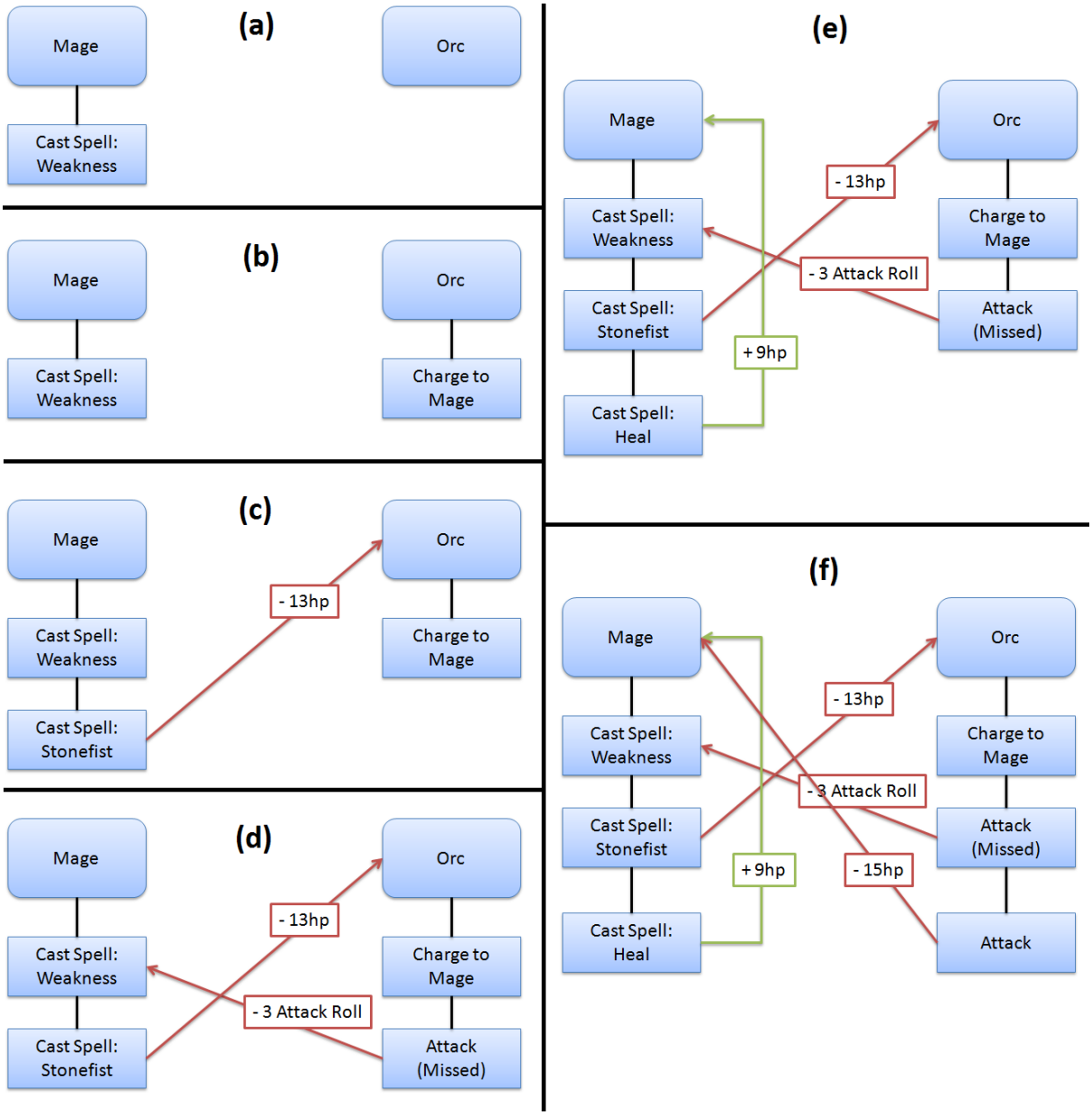

Figure 18: Step-by-step combat representation

a spell called *weakness* over the orc (a). This spell gives a penalty to the next attack roll. Because this action (spell *weakness*) generates an influence (in this case, a negative influence), it is necessary to save a pointer to this action to be used when the orc makes an attack action. In the orc's turn, due to the distance between him and the mage, the orc can't make an attack action at the current turn, so he runs in the direction of the mage to put him in melee range (b).

On the next turn, the mage cast another spell (*stonefist*) (c), which only causes thirteen of damage to the orc's HP. This damage is considered an influence, thus the action connects to the one influenced by it, as can be seen by the red arrow connecting the action with the orc. In the orc's turn, the orc makes an attack action because he is now in melee range of the mage (d). However, due to the spell casted by the mage (*weakness*) in the last round, the orc suffers a penalty to his attack roll and misses the attack. Because his attack was influenced by the spell, then the attack also have a connection to the spell, as can be seen by the red arrow connecting both actions. The label indicates the type of influence the attack action received, which is a penalty of three points in the attack roll, reducing the probability of scoring a hit.

In the following turn, the mage casts a spell (*heal*) on himself, which removes the damage taken[7] (f). The healing influenced (positively) the mage, thus the healing action has a direct connection to the mage expressing the influence (+9hp). This influence is represented by the green arrow. Then, in the orc's turn, the orc attacks the mage (f). This time, he scores a hit, dealing damage to the mage. Since the action caused damage to the mage, it has a direct connection to the mage because it influenced the mage's HP. This connection is represented by the red arrow linking "attack" and "mage" with the label "-15hp" expressing the damage done.

Agents present in a scene, or place, are also represented by lists. Each scene has a list of *entities* and *agents* that are in it, which, in turn, contains a list of actions executed while in the scene. Since the sequence of actions in a game is important, situations where the same place is visited multiple times throughout the game, like a city, might cause complications in graph, making it difficult to understand. To avoid this issue, instead of putting everything in the same scene, each visit to the place is treated as a different instance. For example, if the player visited a city then went to an adventure in a nearby forest and later came back to the city, instead of grouping all actions from both times the player visited the city, each visit is treated as different places, or instances. This is reasonable because even if it is the same city,

[7] In the example, the mage only took damage at the second attack from the orc (the first attack missed). However, that does not mean that the mage entered combat with full HP, thus the healing spell.

it was visited at different times and might have different aspects. This approach results in a clearer visualization of the player's journey and interactions and the sequence of places the player visited.

All collected information from the game composes the *game flux log* that is used for the generation of a provenance graph. However, using the data structure presented in the previous paragraphs, the *game flux log* might still be huge, increasing the size of the provenance graph. To reduce the graph's size, it is possible to make inferences thus omitting or hiding some information for a better analysis. However, all information present in the graph is preserved even when inferences are made. An inference only omits information and does not remove them from the graph. So, instead of recording everything in the game, deciding which information should be stored might be useful for reducing the provenance graph size and, depending on how the filtering is done, no relevant information from the game will be lost.

For example, the number of *agents* present in a village can range from hundreds to a few thousands. So, instead of collecting information from all *agents*, knowing that almost all are making similar movements, it can be collected only from the ones that interacted with or influenced the actions of other *agents* relevant to the story. By doing this, we filter *agents* that are only there, for instance, to simulate crowd (PASSOS *et al.*, 2009). Another possible filter is for actions. Sitting in a bench, opening a window, or jumping around while walking can be filtered depending on the context in which they were executed. Filtering these types of non-essential actions or *agents* decreases the quantity of gathered information, which in turn reduces the size of the provenance graph that is generated later.

This filtering can also be done after the *game flux log* was generated and before it is used for the provenance graph. It can also be done in both stages, while the game is running and after the log is generated. When the game session is running, minor filters can be used to reduce the *game flux log* size. When the session is over, it is also possible to apply other types of filters to reduce even more the size of the log. The more irrelevant information removed in this stage, the fewer inferences will be required during the graph visualization in order to clear the graph from unnecessary information. This way, the user is able to devote more of his attention to analyze relevant data.

4.3 PROVENANCE VISUALIZATION

The purpose of collecting information during a game session is to be able to generate a provenance graph and use provenance techniques in order to analyze and infer the reasons of the outcome. The provenance gathering stage that stores such information was introduced in

the previous section. However, not all stored information in the *game flux log* is relevant for the analysis, even when pre-filtering the information before processing the graph. These irrelevant elements act as noise and can be omitted by inferences during some provenance analysis, since some actions might not be relevant for one type of analysis but essential for another type.

This section introduces the provenance visualization stage, which allows for a visual representation of the generated *game flux log* through a provenance graph. A game using the *Provenance in Games* framework is able to generate a *game flux log*, as described by the provenance gathering stage in Section 4.2, and display a provenance graph that represents the *game flux log* at the provenance visualization stage. This provenance graph allows the user to visually analyze the gathered data.

At the end of the game session, or at any moment during it, the *game flux log* is generated containing all collected information of the session until that moment of the provenance gathering stage. This log is then processed and used to generate a provenance graph at the provenance visualization stage. The graph construction is based on the information contained in the log. The graph itself is a visual representation of the *game flux log*, allowing the user to interact and analyze the information collected from the game session. This aids the user in understanding how the events in the game occurred and how they affected the outcome. The graph also allows for the visualization of the consequences of each action, if any, on other elements in the game, either directly or indirectly.

The construction of the graph is based on a set of rules that are used to interpret the information in the *game flux log*. The information is extracted from the log and used to create the respective visual representations in the graph, motivating the creation of vertices and edges. The vertices of the graph represent *activities*, *entities*, and *agents* present in the game, whereas edges represent their relationships, which are influences or associations. Direct influences are easily spotted by their corresponding edges. However, indirect influences might require some inferences until the user can visually identify them. For instance, collapsing chains of actions can highlight find indirect influences. Moreover, omitting facts can also be used to remove unnecessary or irrelevant information that came with the *game flux log*, allowing a better understanding and clearer visualization of what is relevant for a specific analysis. No information is lost in this process, so it is possible to undo all changes made during the process. The following sub-sections detail features for displaying information in the provenance graph, as well as available types of filters.

4.3.1 VERTEX CHARACTERIZATION

Because the provenance visualization stage uses a visual representation (provenance graph) of the *game flux log*, certain features are used to aid the visualization and distinction of the information displayed from the *game flux log*. One of such features is the vertex shape. Other features include the usage of colors and borders to distinguish displayed information according to their relevance and impact. These features use the information contained in the vertices and edges to determine their visual attributes. It is also possible to use labels to express some information. For example, vertices can show their timestamps and names as labels while edges can show their type of influence (ex: damage, healing, buff) as labels.

As previously noticed, vertices can have different shapes according to their types. *Activities* are represented as squares, *entities* as circles, and *agents* as hexagons. However, it is also possible to differentiate a vertex from another by using different borders as well as colors. As an example, *activities* that do not interact with other *activities* or entities are dashed, as illustrated in upper right corner of Figure 19. Also in Figure 19, activities are colored as light gray, agents are dark blue, and *entities* are beige and dark gray. Color can also be used to distinguish *agents*, *activities*, and *entities* according to their relevance or sub-type. For example, the beige and dark gray *entities* in Figure 19 illustrates that they have different types. This is useful because it is possible to distinguish a player from enemies by using different colors since both types are *agents*, thus have the same shape.

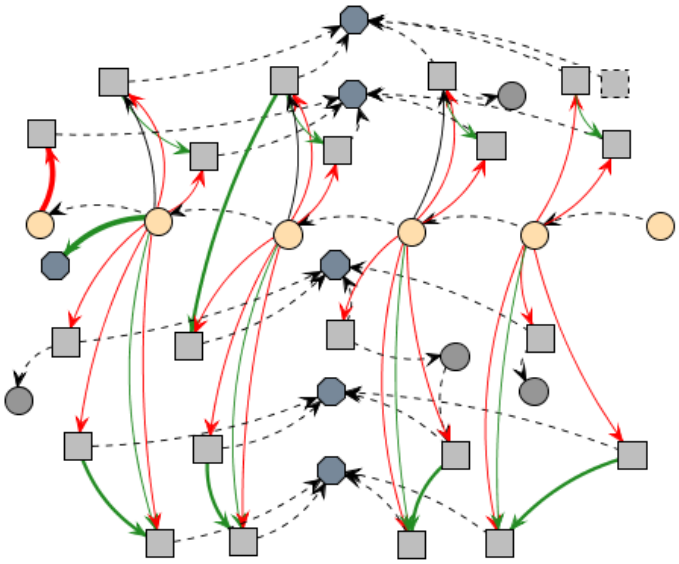

Figure 19: Example of a generated provenance graph

Different formats can also be used for edges, as well as colors. The thickness can be interpreted as how strong the relationship is. If the edge represents a low influence on the *activity*, it is drawn as a thin edge. If the influence is high, then it becomes thicker. This

feature can be used to quickly identify strong influences in the graph just by looking at the edge's thickness. The edge's color is used to represent the type of relationship, which can be any of these three types: positive, which indicates a beneficial relation; negative, which is a prejudicial relation; and neutral, which is neither beneficial nor prejudicial. For each type of relationship (positive, negative, and neutral) a different color is used. Green is used for positive influences, red for negative, and black for neutral. Lastly, dashed edges represent edges without values, which are association edges such as the edges binding activities to an agent. These edge types are also illustrated in Figure 19, where neutral edges are dashed to emphasize their lack of importance.

Despite vertices that represent *activities*, *entities*, and *agents*, it is also possible to create other types of vertex for the graph in order to better organize it. For example, it is possible to create a vertex type to represent locations and bind all actions that took place in each location, as well as instances of the agents that were also in the location. Moreover, the player's journey could be represented by linking each location according to order it was visited.

4.3.2 FILTERS

Since the graph is generated from collected game data, not all collected information is relevant for every type of analysis. Thus, the provenance graph might contain actions that did not provoke any significant change or are not relevant for the desired analysis. These elements act as noise and can be omitted from the provenance graph during analysis through filters. These filters can be of two types: vertex filter and edge filter. These filters are related to the graph elements themselves, omitting vertices and edges. Another feature present is the attribute status display, which alters the way the information is displayed. For example, to better analyze the HP attribute, both from monsters and players, the attribute status display changes the colors of all vertices that contain such attribute while keeping all other vertices intact.

Filters can also be used to collapse vertices in order to reduce the graph size by changing the information display scale, grouping nearby vertices. For example, instead of displaying information in a daily basis, it is possible to group together each 7 days of information in order to display the summary of the events in a week scale. Another usage of collapse is to group *activities* from the same *agent*, making easier to see all influences of the *agent* throughout the game. Similar to the vertex filter, the edge filter is used to omit information, in this case, relationships between vertices by types of relationships. One

example is to filter all edges that express damage done during the game, thus omitting such edges at the graph.

The last feature present is the attribute status display. When selecting the desired attribute, all vertices with the specified status will have their colors changed according to their respective values. It uses the traffic light scale (DIEHL, 2007), which indicates the status of the variable using red (*i.e.* below 40%), yellow (*i.e.* between 40% and 75%), or green color (*i.e.* greater than 75%). These thresholds are customizable to each type of status. As an example, imagine that it is desired to analyze the player's HP value throughout the game. When filtered by player's HP, all vertices that contain a player HP value will have their colors changed according to its value. Activating this type of filter allows the user to quickly check the player's HP throughout the game, making it easier to identify situations where he might have had trouble, which is distinguished by red color. Chapter 5 provides examples of these features.

4.4 FINAL CONSIDERATIONS

This chapter presented a conceptual framework to gather detailed gameplay information for visual analysis by the means of a provenance graph. The conceptual framework, called *Provenance in Games*, provides the necessary mappings of provenance terms into the game elements and is divided in two stages: provenance gathering and provenance visualization. The provenance gathering stage is responsible gathering gameplay data and uses this data to create the *game flux log*. The second stage, provenance visualization, is responsible for generating the provenance graph from a *game flux log* to visually represent the gathered gameplay information. The provenance graph allows a better understanding of the events occurred during a game session by allowing the user to visually identify the influences in the game.

Currently, Proof Viewer does not provide inference for the user, only the necessary means to infer. The game developers need to create inference rules customized to their games, such as clustering sequences of actions and identify irrelevant sections that can be omitted from the user. Providing a generic inference strategy is a future work. To infer something and decide if it is relevant or not for analysis is a complex process, which happens to be domain specific. This type of decision making also involve other areas of research (BRISTOL, 1977; CIOS *et al.*, 1998; FAYYAD *et al.*, 1996; HAN; KAMBER, 2006; WITTEN; FRANK, 2005) and varies from games to games.

The next chapter presents a game that used the *Provenance in Games* conceptual framework to generate a *game flux log* and a provenance graph. The log is used to generate the provenance graph and visually represent the game session, while also giving examples of possible analysis. It also details implementation aspects of our approach.

CHAPTER 5 – IMPLEMENTATION

5.1 INTRODUCTION

The game flux analysis deserves particular attention for serious games (ABT, 1987), which are games used for purposes that go beyond entertainment. Serious games have been used for aiding students to learn and understand concepts taught in classrooms (BAKER *et al.*, 2003; KOHWALTER *et al.*, 2011; NAVARRO; VAN DER HOEK, 2004) due to their characteristic of stimulating curiosity and providing motivation for learning (PRENSKY, 2001) or within simulation of situations and scenarios, providing resources for training. Commonly, understanding the educational results obtained in a serious game are important to assimilate the knowledge and concepts passed in the game. In addition, examining the game flux allows for the identification of good and bad attitudes made by the player or by game developers. This knowledge can be used in future game sessions to avoid making the same mistakes or even to adjust gameplay features.

In this chapter, the *Provenance in Games* conceptual framework is applied in the *Software Development Manager* (SDM) game (KOHWALTER *et al.*, 2011) as a proof of concept. The SDM game focuses on introducing Software Engineering concepts. The version of SDM presented in this chapter makes use of the conceptual framework for collecting provenance and generating the *game flux log*. The generated log can be viewed by using *Prov Viewer*, a tool from our approach that displays the provenance graph. While we developed *Prov Viewer* as a customized tool, compatible with SDM, it can support other games with few modifications on the interface, filters, and vertices.

This chapter is organized as follows: Section 5.2 briefly describes the SDM and Section 5.3 details of how the provenance information is gathered, generating the *game flux log*. Section 5.4 provides a simple example of game session in SDM. Section 5.5 describes details about *Prov Viewer*. Lastly, Section 5.6 presents the final considerations of this chapter.

5.2 SDM

In SDM, which was developed using the game engine Unity3D (HIGGINS, 2010), the player has a team of employees that are used to develop software according to contracts made with customers. The gameplay and game mechanics are modeled presenting possibilities to the player, which decides strategies for development and defines the roles for each staff member. As in any contract, the software has requirements that must be followed during

development. From a gameplay point of view, these requirements help to balance the mechanics and rules. When the software is completed and delivered to the customer, there is a quality assessment of the software and a project completion payment accordingly to the product quality. Figure 20 presents a screenshot of SDM in action, with the bottom corner illustrating the software's development status.

Figure 20: Screenshot from a game session in SDM

Since SDM focuses in people management, the main characters of the game are the employees, which represent the player's labor force. Employees can perform different roles (analyst, architect, manager, marketing, programmer, and tester), which use the employees' attributes to calculate their performance depending on the respective roles. Their names and roles are displayed at the top corner of Figure 20. Another element present in the game is specialization, used to define the employee working competence. With the specialization system, it is possible for employees to undergo training to learn new sets of skills. Also the concepts of working hours, morale, and stamina are used to modify the employee's productivity. The left corner of Figure 20 illustrates the status of morale and stamina for each employee in the staff.

The gameplay is based on resources administration, where the player must control the software production based on the requirements established by the client. The player has to make decisions related to his staff, which varies from one to eight employees, regarding the development of the software. These decisions can be of tasks and role designations, placing employees in training, and staff management (hiring, firing, working hours, and promotions).

Figure 21 shows a simplified version of SDM's class diagram focusing on the employee, displaying his human attributes, types of specializations, the possibility of training to acquire specializations, and that the employee is affected by the other employees in the staff team. The yellow classes represent generic classes from the conceptual framework presented in Figure 16. It also illustrates the project, its characteristics and requirement. The next subsections describe how the information is stored in the game and show examples of analysis of the generated provenance graph.

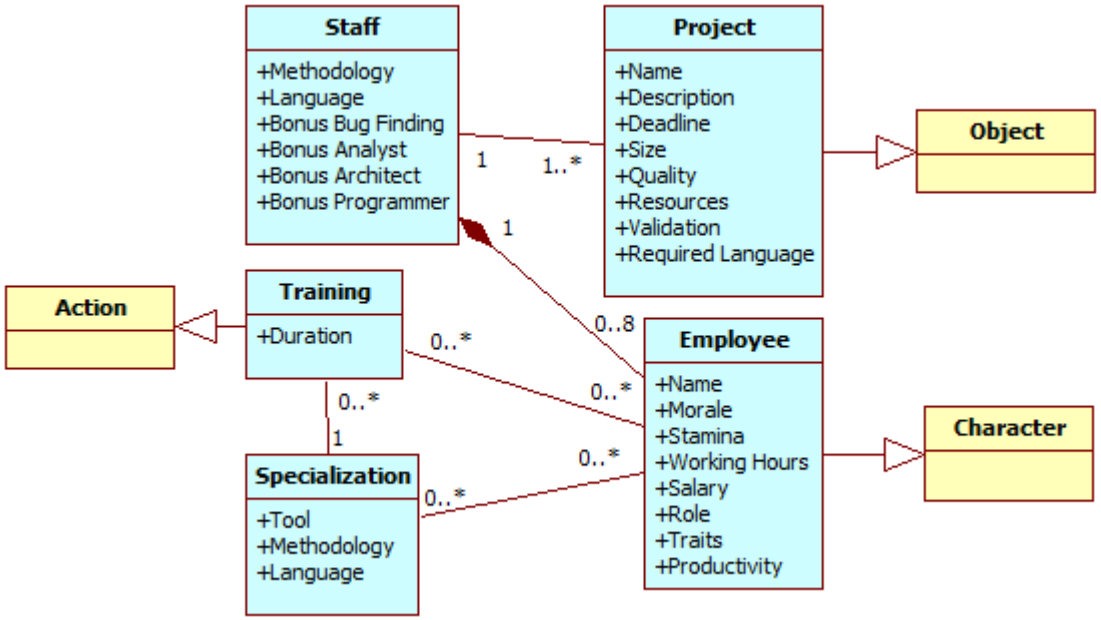

Figure 21: SDM simplified class diagram.

5.3 PROVENANCE GATHERING

The data structure used in SDM was adapted and mapped to be suitable with the proposed provenance structure explained at chapter 4. Each project contains a list of the employees involved in its development. In turn, each employee has a list of his actions executed throughout the development. If any action had an external influence during its execution, then the action also has a pointer to the action that influenced it. Throughout the game, when actions are executed or new employees are hired, information about the event is collected and stored for later usage. The actions go to their respective lists while new

employees are added to the project list, creating their own list of actions. Each day passed in the game also records the state of the software development at the end of the day.

Since the information collected is used for the generation of the provenance graph, its content is mapped to the three possible types of provenance vertex: activities, agents, and entities. This mapping is done according to the data model explained at chapter 4. The following paragraphs describe information details that are extracted from the game and their respective roles in the provenance graph.

Each action executed during the game is represented by an *activity* vertex. The information collected during its execution includes: who executed it, which task and role the employee was occupying, and the current morale and stamina status of the employee that executed the action. The worked hours in the day the action was generated and credits spent to execute the action are also stored, along with the progress the employee made during his task. These elements are illustrated in Figure 22, detailing the *entity* vertex representation for the project's data (a), the employee's *agent* vertex data (b), and the action's *activity* vertex data (c). Besides those, if the action had any external influences, such as the use of an artifact (prototypes or test cases, for example), then SDM stores a link to the action or artifact that affected its execution.

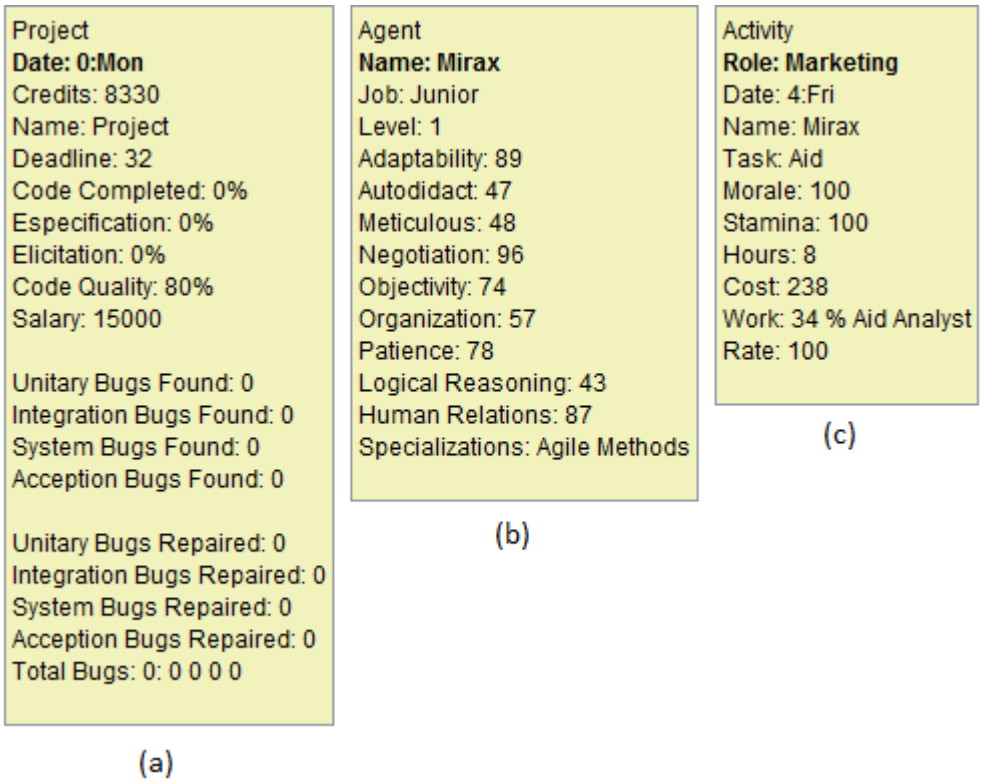

Figure 22: Information data extracted and visible at the provenance graph.

Each employee that participated in the development of the software is mapped to an *agent* vertex in the provenance graph. The collected information includes the employee's name, his current staff grade (junior, mid-level, or senior), his current level (and experience points), traits, and specializations. Lastly, the *entity* vertex in the provenance graph represents one of the three possible artifacts in SDM: Prototypes produced by architects and used by analysts; Test Cases produced by analysts, architects, and programmers and used by testers; and Project, which represents the instances of the software development progress recorded each day.

The daily project information collected includes the day of its instance, the project's deadline, how much coding was produced and the code overall quality. It also stores the clients requirements identified and modeled by analysts, how many credits the player had by the end of that day, and the state of each type of bugs in the software. For prototypes and test cases, only the day they were created and their names are stored, since actions contain the information of when they were used.

Figure 22 illustrates the information collected in SDM and shown in *Prov Viewer* according to the vertex's type. For a Project vertex (*Entity*), the state of development of the software is daily collected, such as how much coding was done, the code quality, how many reported bugs (found) and how many were repaired. For an Agent vertex (*Agent*), it collects the character's attributes, current level, his current placement in the company (job), and his specializations. For a Process vertex (*Activity*), it collects the character that executed the activity, his morale and stamina status, the task performed, number of worked hours, the day the activity was executed, how much it cost the player, and the outcome of the activity (work). For example, Mirax at day 4 had the "aid" task and generated a +34% bonus to all analysts that worked that day.

At the end of a gaming session, the data collected during the game session, also known as *game flux log*, is exported to an external visualization tool, the *Prov Viewer*. *Prov Viewer*, in turn, processes the data and generates the corresponding provenance graph that represents the game session. The next section describes a game scenario in SDM that is used as an example for describing the features present in *Prov Viewer*.

5.4 GUIDING EXAMPLE

In this section, we describe an example of a SDM game session, which explores the development process of software in the game. Over the span of five in-game weeks, the player makes various decisions that directly affect the software development, such as hiring new

employees, designating roles and tasks, and modifying the work hours of each employee. The video from this game session is also available at GEMS[8].

Starting the game, the player has at his disposal four employees: Yesha, Tornik, Mirax, and Emmy. The first thing he does is to assign roles for each employee. Yesha is assigned as the staff's manager and has the task of aiding analysts. Tornik is assigned as an analyst, Mirax as marketing (which aids analysts and provides a cash income to the player by making deals), and, lastly, Emmy is assigned as programmer to develop the software. Then the player asks Yesha to hire tree new employees: Arden, which is placed in training, Marke, an architect, and lastly Daniel, an analyst that will work for 14 hours a day. Almost two weeks passed before Arden finished his training and was allocated to work as programmer.

Starting the third week in the game, the player begins to have financial problems. He is running out of cash. Daniel, due to the extra hours, is tired and later quits. The game continues with a few rearrangements in task. Tornik is assigned to do both elicitation and specification tasks as analyst and Arden begins to work as a programmer. Mirax is later promoted at the third week. At the fourth week, Marke's role is changed to programmer, focusing on repairing reported bugs, and as a tester. Nearing the end of the week, Arden and Marke resign the staff due to lack of payments since the player was having financial problems. At the start of the next month, and after receiving cash from achieving a milestone from the contract with the client, the player hires another employee (Miera) as a programmer to replace Arden. At the same week, the player sets Mirax to negotiate with the client, asking to extend the project's deadline by one extra week, since the deadline was ending. Because of the deadline extension, the staff manages to complete the software in time, delivering the software to the client.

The software delivered still had one reported and unfixed bug, plus another twenty five unknown bugs that were not identified by the staff. Aside from the bugs, the coding quality of the software was mediocre with a rate of 75.84. This rate can vary from 10 to 120, where 10 is the maximum negative modifier and above 100 provide a positive modifier. Thus 75.84 is near the average (65.0). Concerning the player's financial status, the player started the game with 40,000 credits and at the end he had 5,969 credits and gained another 8,335 credits (out of 34,335) for delivering the software. The difference in payment is due to the number of bugs left in the software (26 bugs). Also, the player's reputation did not increase because of the poor quality of the delivered software (number of bugs). Concerning the staff,

[8] The video can be downloaded at http://gems.ic.uff.br/ping/.

the player kept all the starting employees, but lost three out of four hired employees. Three of the remaining employees have lost morale during the development and one is fatigued.

At the end of the session, the *game flux log* was generated by using the collected information from the game (employees, actions, and the project daily progression). The examples and illustrations presented at the next section are based on the *game flux log* exported from the game session described in this section.

5.5 PROV VIEWER

The *game flux log* is used by *Prov Viewer* to generate a provenance graph corresponding to the game session. In order to do this, *Prov Viewer* processes the information and interprets it to generate the vertices and edges of the graph, as illustrated by Figure 23. As can be seen, the process is divided by three phases: Processing the *game flux log*, generating the graph, and drawing the graph.

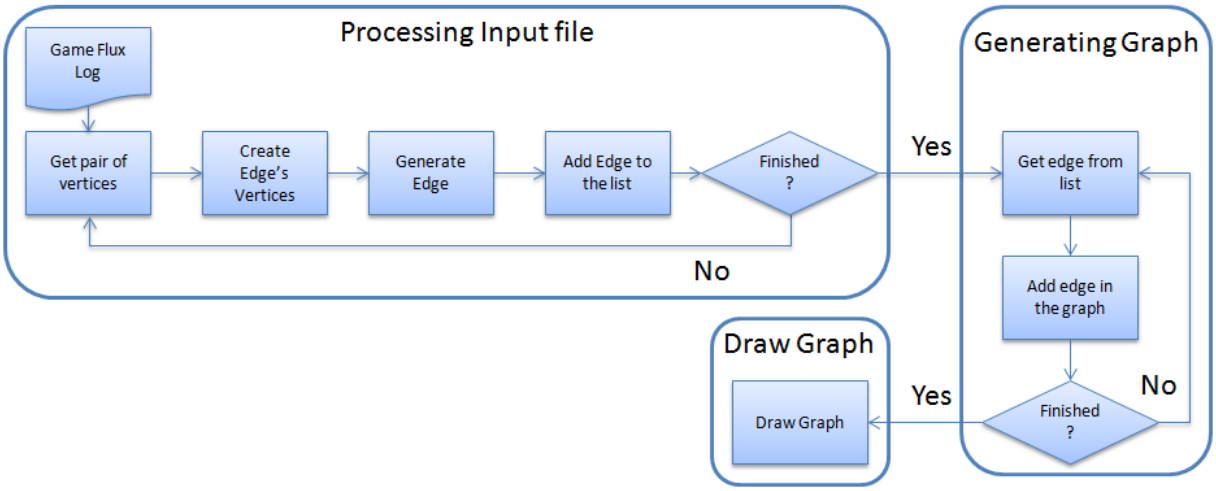

Figure 23: *Prov Viewer* processing the *game flux log* and generating the graph.

Firstly, the *game flux log* is processed, classifying the information to their corresponding vertex types (*activity*, *entity*, or *agent*), and generating the edges that link each vertex in the graph. To simplify this step, the information extracted from the game is arranged in pairs, where each pair represents two vertices followed by the edge that links them. As such, *Prov Viewer* generates the vertices and edge every time it processes a pair of vertices. Each time *Prov Viewer* process a vertex, it searches the database to check if the vertex was already processed. If so, *Prov Viewer* uses the processed vertex instead of creating a new entry. Otherwise, it creates the vertex. This avoids duplications in this step, since a single vertex can appear multiple times in the *game flux log* due to the nature of how the *log* is structured: a vertex, another vertex, and the edge that links them. Thus, the vertex would appear multiple times in the log if it had multiple edges connecting it.

After processing both vertices, *Prov Viewer* creates the edge and stores it in a list of edges that is later used to generate the graph. In *Prov Viewer*, an edge contains pointers to the source vertex, target vertex, and edge's information (value and type). The source and target are the previously processed vertices from the pair. This is done until the entire *game flux log* is processed and all edges are placed in a list of edges that is used to generate the graph. All information from the *game flux log* is processed in this stage, even if they don't initially appear in the graph, due to filters.

As mentioned earlier, vertices can belong to three types: *activities*, *entities*, and *agents*. When generating the *game flux log* that contains the information extracted from the game, an additional tag is added to distinguish the vertices types. *Prov Viewer* uses this tag for generating the vertex instead of deciding which vertex type it belongs to according to the vertex characteristics, thus saving processing time. Note however that the input format can be customized, as long as it generates a list of edges, where each element in the list has the vertex source, the vertex target, and the edge information. The structure used for each line in the text file is composed of: Tag + Vertex + Vertex + Edge.

The next step is the generation of the graph. *Prov Viewer* uses the generated list of edges, creating each edge in the graph, and, consequentially, the vertices from the edges. It is done this way because *Prov Viewer* uses the JUNG framework (JOSHUA O'MADADHAIN *et al.*, 2010), where an edge is created by adding the edge in the graph from *source* to *target* with the information *edge*. If *source* and/or *target* are not in the graph, then JUNG automatically generates the vertex. This avoids the need of creating each vertex before creating the edge in the graph, while at the same time, checking for duplicates. After creating the graph, it is drawn on the screen and displayed to the user.

5.5.1 GRAPH VISUALIZATION AND REPRESENTATIONS

After the *game flux log* is processed and the graph is generated, it is drawn on the screen so the user can analyze it. Figure 24 illustrates the graphical user interface (GUI) of *Prov Viewer*, using the provenance graph generated from the scenario discussed in section 5.4. Square vertex represents an activity, while circle represents an entity and an octagon represents an agent. The provenance graph is displayed at the center of the screen but only a part of it is visible due to the graph size. However it is possible to zoom in and out and navigate through the graph. The graph layout is set to be similar to a spread sheet, were each "line" represents the activities of each agent and each "column" represents a day in the game. The layout can be customized by creating new layouts or using existing ones available from

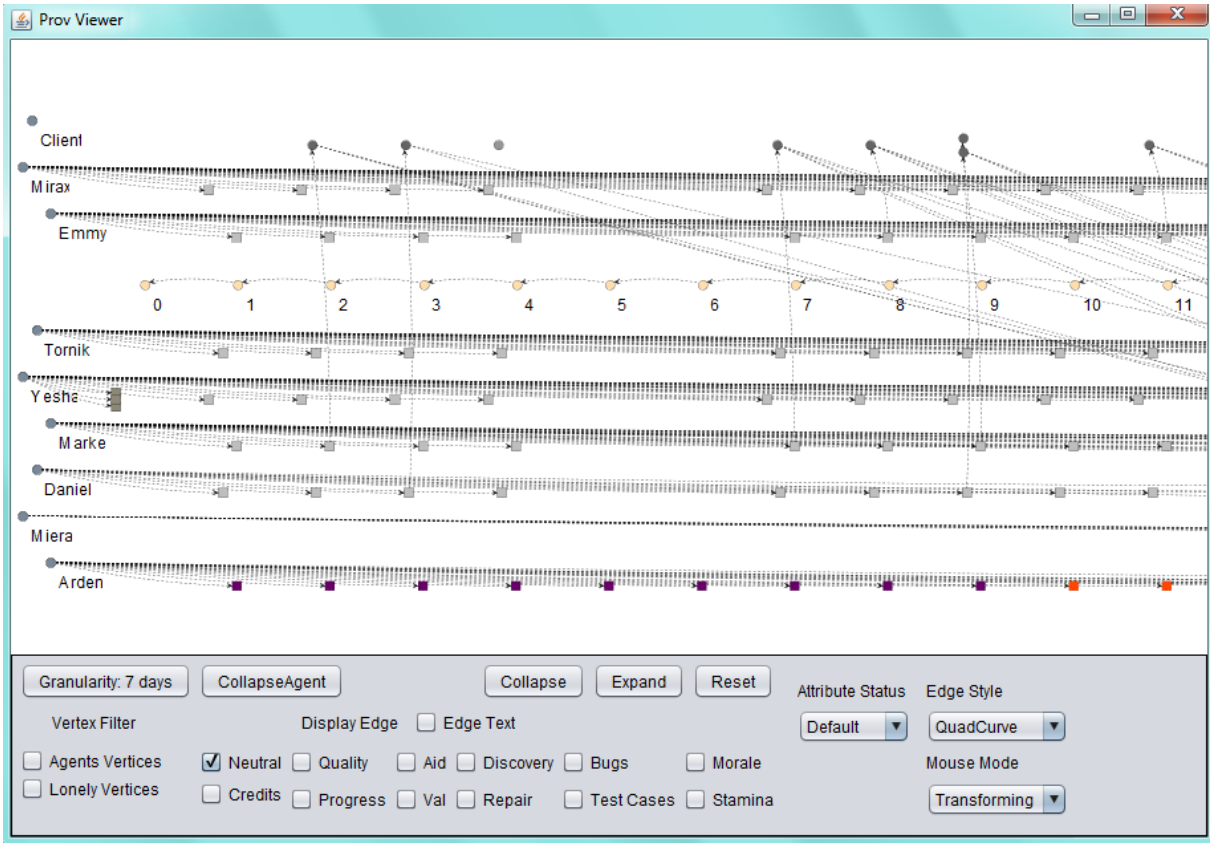

Figure 24: Prov Viewer's GUI

JUNG. The filters, which are customized for SDM, are located at the lower region of the interface. Starting with the buttons, the first one is "Granularity: 7 days". This button is only an example of grouping vertices together for the same agent. In this case, it groups vertices from the same week. This is useful for huge graphs, which allows summarizing displayed information in a weekly basis. Figure 25 illustrates the graph after turning the granularity feature on, grouping vertices in the graph for each week.

The "CollapseAgent" button collapses all the agent's vertices into the agent itself. It can be useful to detect if an agent had any influence throughout the game, instead of looking vertex by vertex. The "Collapse" button allows the user to collapse selected vertices, while the "Extend" button remove the last collapse made to generate the selected vertex. Figure 26 illustrates an example of "CollapseAgent" and "Collapse" features while showing neutral and aid edges types. Using "CollapseAgent" on Mirax allows for easy identification that she had influenced other characters. The same figure illustrates a collapse of Daniel's second week activities at the bottom right corner.

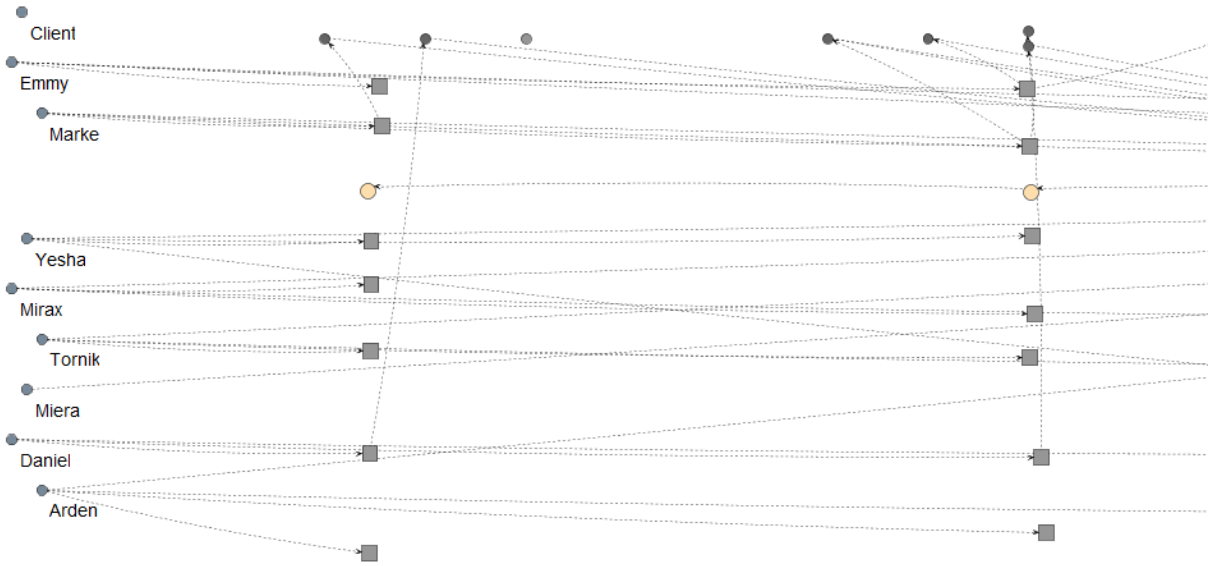

Figure 25: Graph from Figure 24 with "Granularity: 7 days"

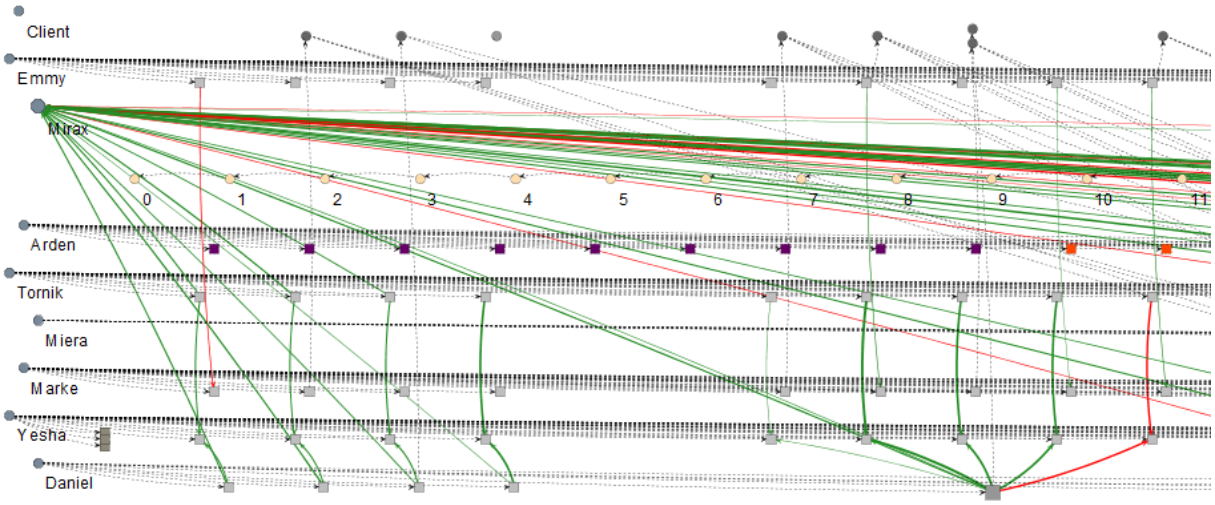

Figure 26: "CollapseAgent" (Mirax) and "Collapse" (Daniel's second week)

The "Reset" button removes all modifications made in the graph, setting it to the original state. The "Edge Style" is used to change the edge's arrow curvature from a quad-curve to straight lines. Lastly, the "Mouse Mode" has the purpose of changing the function that the mouse will perform. There are two functions: Transform, which is used to navigate the graph, and Picking, which is used to select vertices in the graph.

The next items in the interface are the checkboxes used for filters. Starting with "Vertex Filters", the "Agents Vertices" hides all agents in the graph. This is just to illustrate the possibility of hiding vertices by type. In this case, it hides *agent* type vertices. The "Lonely Vertices" hides all vertices that have no edge linking them to other vertices in the current displayed graph. This is useful to clean the graph from vertices that have no

edges/influences from the selected types being displayed, reducing the number of vertices on the screen. The “Display Edge” sets the displayed graph to display only the selected types of edges. This is done changing the display status of each edge in the graph, displaying only the types selected while hiding the rest. No information is lost in this process. The information is only hidden from the user. An edge is composed of a value and a type. For example, and edge labeled as *342 validation* has a value of *342* and the *validation* type. In the case of SDM, the edge’s types can be: Credits, Quality, Progress, Aid, Val (short for Validation), Discovery, Repair, Bugs, Test Cases, Morale, and Stamina. Neutral edges are edges that represent association between vertices, while the others represent influences. Black edges represent neutral edges, which are also dotted if the edge’s value equals zero.

Note that only the “Neutral” type is selected in the displayed graph shown in Figure 24. This means that the graph is showing only neutral edges. The graph is set to always start the visualization with only the neutral type selected, pre-filtering all other edges. This is just an example of possible pre-filtering. Any type of filter can be used during the initialization of the graph. This is useful to reduce the graph granularity, hiding information from the user to avoid overwhelming it. The full graph can be seen if all edge’s types are selected, resulting in the section of the graph illustrated by Figure 27. The “Edge Text”, when selected, displays the edge label, containing its value and type. This information is also shown as a tooltip when moving the cursor to the edge. Vertices details are also available by moving the cursor over it.

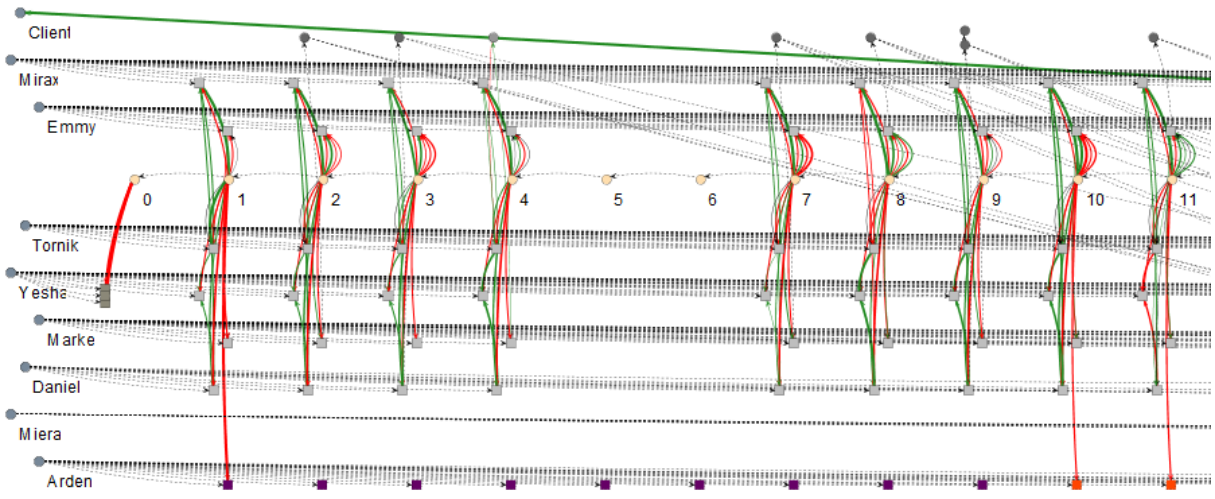

Figure 27: Graph from Figure 24 but with all edges

The edge filter is important because it allows for the identification of types of influences in the graph, filtering other influences that are not relevant for the desired analysis. For example, at days 10 and 11, the employee Daniel had drastic changes in his performance, dropping from *342 validation* to *34 validation*, as shown in Figure 28. The left picture has the

"Val" edge filter on, while the right picture has the "Aid" edge filter on. The employees in the figure are Yesha (upper tasks), Daniel (Middle), and Mirax (bottom). This was detected by activating the “Val” edge filter. The reason for this sudden drop can be traced to Mirax and Yesha by changing the filter to “Aid”. Yesha provided an aid of 298% in day 10 and a penalty of 248% at day 11 to Daniel due to wrong decision making. Moreover, Mirax provided 227% and 136%, respectively for days 10 and 11. By combining these factors, at day 10 Daniel receive a bonus of 525% in his task, while at day 11 he had in total a penalty of 112% for his task. Thoroughly, Daniel productivity without any bonus was 65 at day 10 and 53 at day 11, which is within his productivity margin.

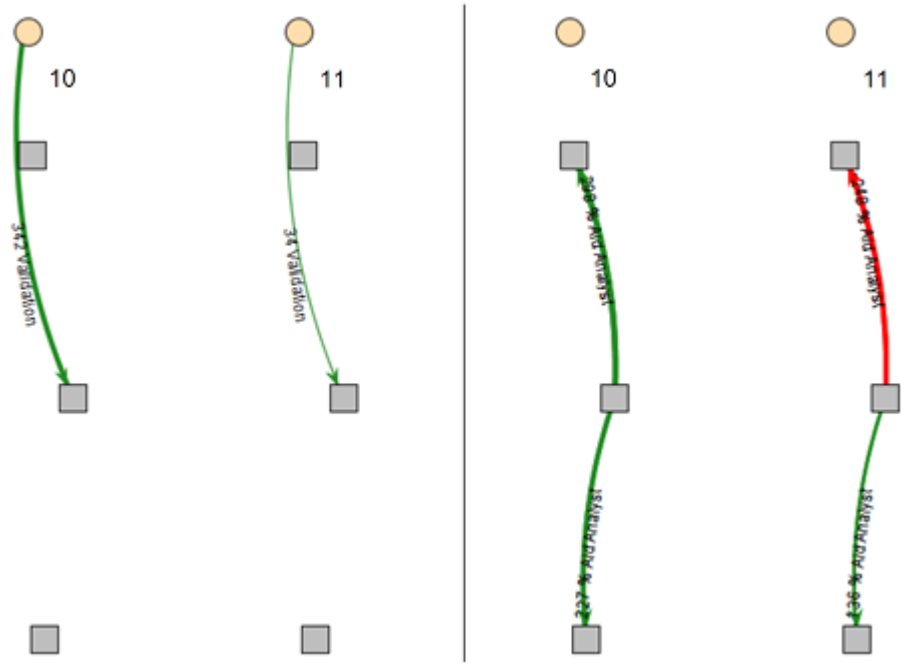

Figure 28: Analyzing Daniel's productivity

The “Attribute Status” changes the vertex color according to their values from the selected attribute. In SDM they can be: Morale, Stamina, Hours (short for Working Hours), Weekend (highlights “Saturday” and “Sunday” vertices), Credits, and Role. The vertex color does not change if it does not have the selected attribute. The default mode colors common activities with a shade of gray and uncommon activities with different colors. Common activities in SDM are normal tasks executed by employees during their roles, while uncommon activities are activities that do not happen frequently. For example, in SDM the uncommon activities are: Idle (red color), Training (purple color), Fired (brown color), Promotion (green color), Hired (“cornsilk” color), and Negotiation (“honeydrew” color). This color difference between vertices is useful to quickly identity non-ordinary events. For example, by looking at the graph shown in Figure 29 it is possible to quickly identify that an

employee trained during one week and was idle during the consecutive four days after the training was complete. In addition, Daniel was fired and Mirax was promoted in the same day. Finally, Yesha hired three new employees at the beginning of the game.

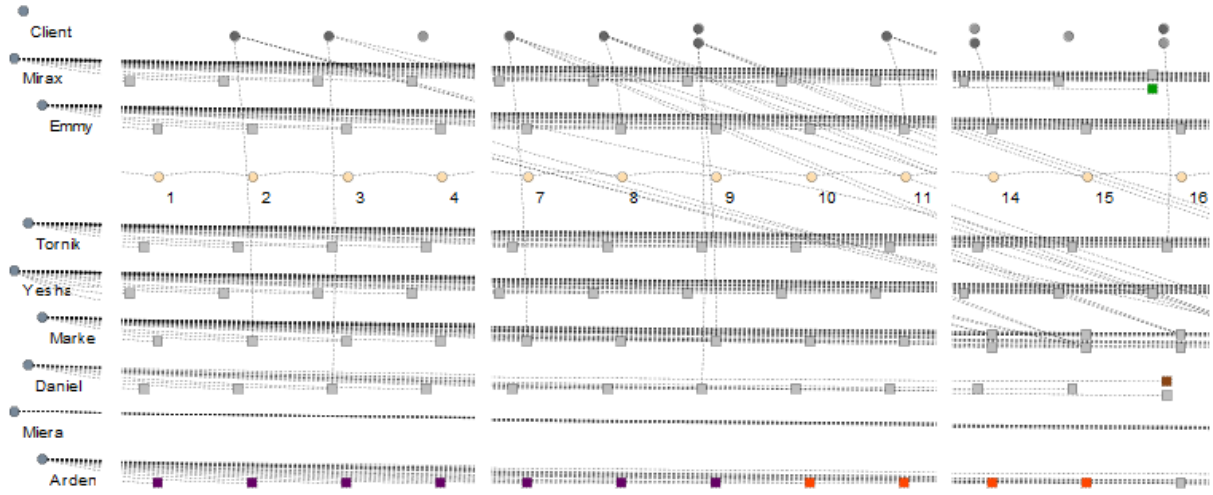

Figure 29: Graph from Figure 24 but focusing on certain sections of the graph

This type of visualization, based on the evaluation of attributes, is useful to quickly identify particular sections in the graph. Another example in the same scenario is to check the player's financial status. By changing the visualization to evaluate Credits instead of the default mode, the vertices that have the player's credits value changes their color according to its status. In SDM, the vertex that contains such information is the Project vertex. By looking at Figure 30, it is possible to see that the player ran out of credits after day 10. It is also possible to identify the source of this problem by activating the "Credits" edge filter, which is illustrated in Figure 31.

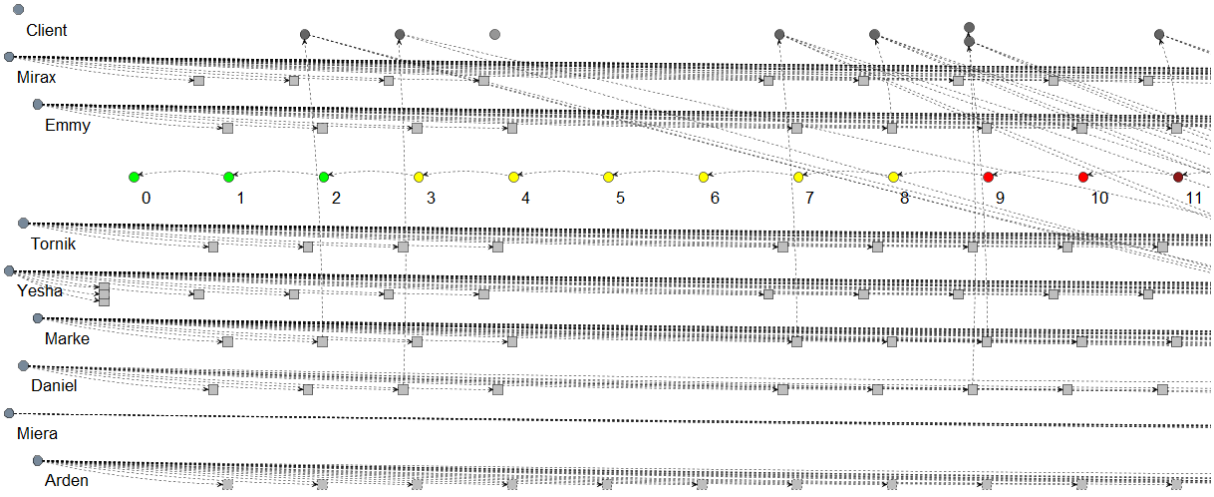

Figure 30: Graph from Figure 24 with Attribute Status set to Credits mode

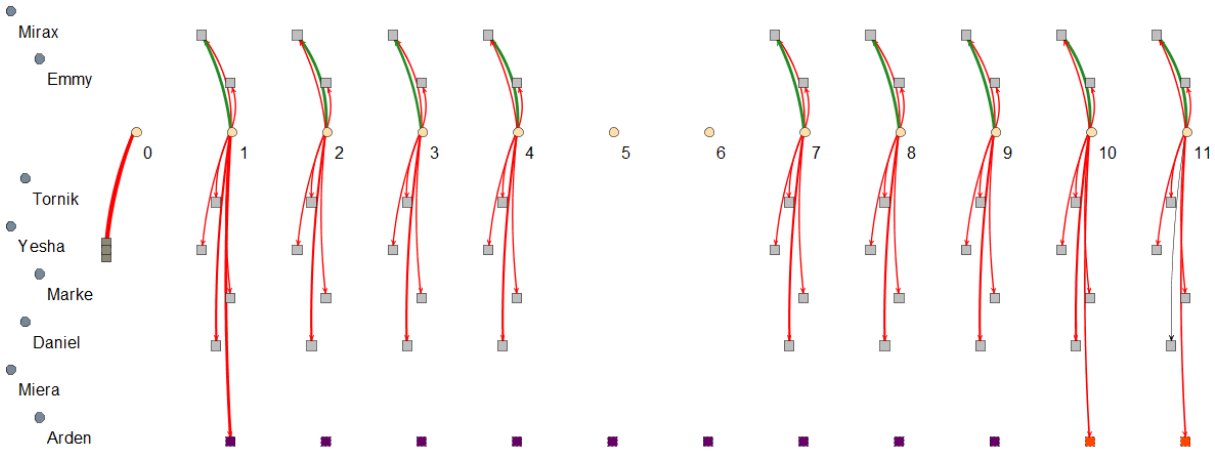

Figure 31: Graph from Figure 24 but with the Credits edge filter on

As can be seen in Figure 31, the player had many expenses with his employees. Hiring three new employees and training another, as illustrated by the thicker edges at days 0 and 1, was a key factor to increase this problem. It is possible to group the vertices together to better visualize the expenses, as illustrated in Figure 32, which grouped these 11 days. This figure was rotated by 90º to the left and the edge's labels were enabled. In total, 24,170 credits were spent with hiring, 8,036 credits with Arden, 0 credits with Miera, because she was not hired until this moment, 3,240 credits with Daniel, 1,971 credits with Marke, 1,899 credits with Yesha, 1,809 credits with Tornick, and 2,007 credits with Emmy. Mirax actually generated 6,840 credits for the player by performing her role as marketing, which provides cash, from side dealings with third parties, and aid analysts.

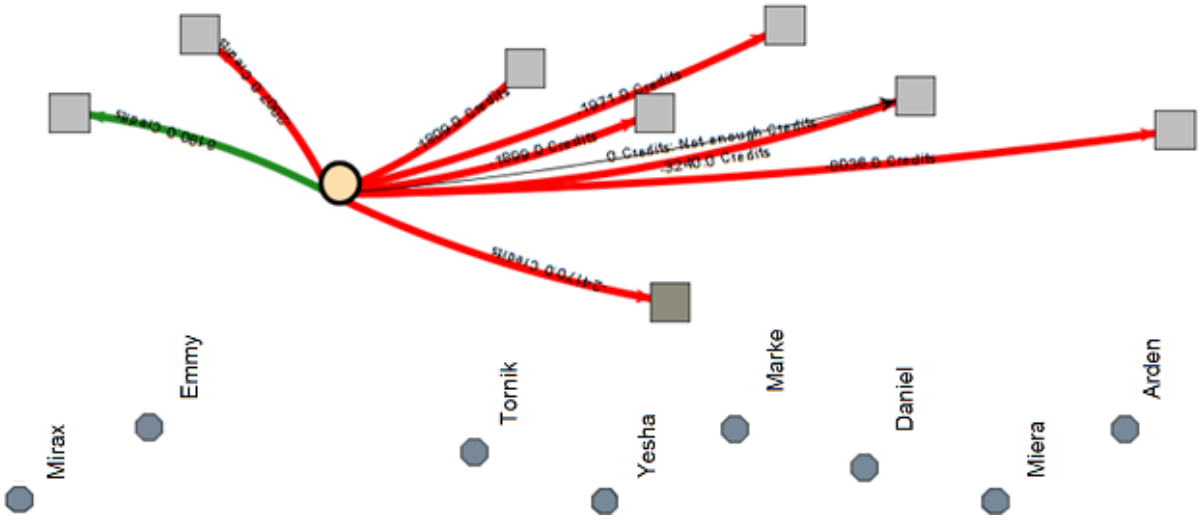

Figure 32: Graph from Figure 24 with the Credit edge filter and collapsed vertices

5.6 FINAL CONSIDERATIONS

In this chapter it was presented the materialization of the conceptual framework presented at chapter 4, encompassing both the collection and visualization. The data gathering

was done in the game SDM, where after each game session the *game flux log* was generated and exported. Then the provenance visualization was done by using *Prov Viewer*, a tool used to generate and display the provenance graph from a *game flux log*. It is important to note that in the example shown in this chapter, we focused only on the issues related to credits and validation influences. However, several other analyzes can be made by changing filters and display views, such as analyzing the reasons that lead employees to quit the staff, which week was more productive and less productive, and the reasons behinds these productivities changes.

Using the conceptual framework present at chapter 4, SDM is able to generate a *game flux log* to be used by the provenance visualization tool *Prov Viewer*, which uses graphs as notation. The contents from the *game flux log* are directly related to the information available in the graph. Features present in *Prov Viewer* were also detailed, such as visualization details and the usage of filters to change the displayed graph.

Even though this chapter shows only one part of the provenance graph (11 days), the original graph consists on 40 days long and contains 273 vertices and was used during the experiments described at chapter 6. However, graphs with more vertices might generate problems for the user in terms of visualization and visual analysis, overwhelming him with information. To deal with this, it is possible to cluster vertices and the edges together in order to simplify the graph. However, currently, these clustering must be done manually or by using the granularity feature. Nevertheless, *Prov Viewer* provides the necessary features to create complex clustering algorithms, such as clustering vertices if they satisfy specific rules or behaviors.

Even though *Prov Viewer* was customized for SDM, it can be trivially adapted to fit in other games. Most resources present in it were designed to work independently of the game context. The few features that are context dependable, such as filters, has templates and is based on abstract classes for easy implementation. Lastly, we did some experiments in order to evaluate the usefulness of this provenance analysis mechanism, as described in chapter 6.

CHAPTER 6 – EVALUATION

6.1 INTRODUCTION

The main motivation of this chapter is related to the research questions defined in Chapter 1:

- Does provenance analysis help to understand events that emerged during the game?
- Is provenance analysis faster than only watching a replay of the game session?
- Is provenance analysis more accurate than only watching a replay of the game session?

To assess the possibility of using provenance analysis for improving understanding, we generated a replay of a game session and compared it with provenance analysis using a provenance graph. This comparison was conducted through a questionnaire containing specific questions about events that occurred during the game session. Volunteers were divided into two groups: with and without provenance. Both groups watched the replay of the game session. The group with provenance also had access to the provenance graph. At the end, both groups answered the questionnaire.

Lastly, we used two metrics to compare the results obtained by both groups: precision and time. The first metric, precision, has the intention to verify the correctness of the answers provided by both groups. The second metric, time, is used to measure the time each volunteer took to answer all questions, thus allowing to know which method (with or without provenance) is faster.

This chapter is organized as follows: Section 6.2 describes details about the experiment planning. Section 6.3 explains the experiment execution, while section 6.4 provides a statistical analysis by detailing tests, their results, and conclusions related to the obtained data. Section 6.5 discusses some threats to validity of the experiment. Lastly, Section 6.6 presents the final considerations of this chapter.

6.2 EXPERIMENT PLANNING

We opted for a controlled environment in order to reduce independent variables that were beyond our control. Instead of playing the game, volunteers watched a recorded game session played by a third person. Thus, the questionnaire can be customized to the game session, allowing asking specific questions about events that occurred in that particular

session. Also, the questionnaire is designed to measure the precision of the answers provided by both groups (with and without provenance) and the time volunteers took to finish it. Precision (BAEZA-YATES; RIBEIRO-NETO, 1999) is a traditional metric for information retrieval and can be seen as a measure of correctness, which is the percentage results that are relevant. Time is measured in minutes taken to complete the questionnaire.

Before filling the questionnaire, volunteers are required to read and watch tutorials due to the unfamiliarity with the game and the *Prov Viewer*. Furthermore, we ran a pilot of the experiment in order to determine the experiment structure, which was initially structured as follows: volunteers were divided into two groups and start the experiment by watching the SDM tutorial, then the *Prov Viewer* tutorial (only for the group with provenance) and the replay of the game session video. Lastly, they receive the questionnaire.

This order was later changed for the experiment due to the fact that volunteers were reviewing the *Prov Viewer* tutorial while answering the questionnaire. This happened because they were forgetting how to operate the tool after watching the replay of the game session video, which takes around seven minutes. Another change made for the experiment was related to the questions in the questionnaire. Some questions were leaving room for different interpretations, which caused too many mistakes on both groups. Thus, we decided to create a new scenario (and video) with a different set of questions. Lastly, during the pilot we allowed each volunteer to watch the videos at their own pace, causing chaos because of undisciplined behavior from the volunteers. They were also deceiving the time they took to answer the questionnaire. Thus, we decided to impose a stricter timetable, providing the questionnaire only after all volunteers of the same group finish watching the videos.

With the changes made after the pilot, the experiment plan is illustrated by Figure 33 and is divided in three stages: Generating the questionnaire, running the experiment with volunteers (students), and analyzing the results. According to the plan shown in Figure 33, the first stage (Create Questionnaire) is executed before running the experiment with volunteers. We created the replay of a recorded game session from SDM that narrates the player’s decisions throughout the game.

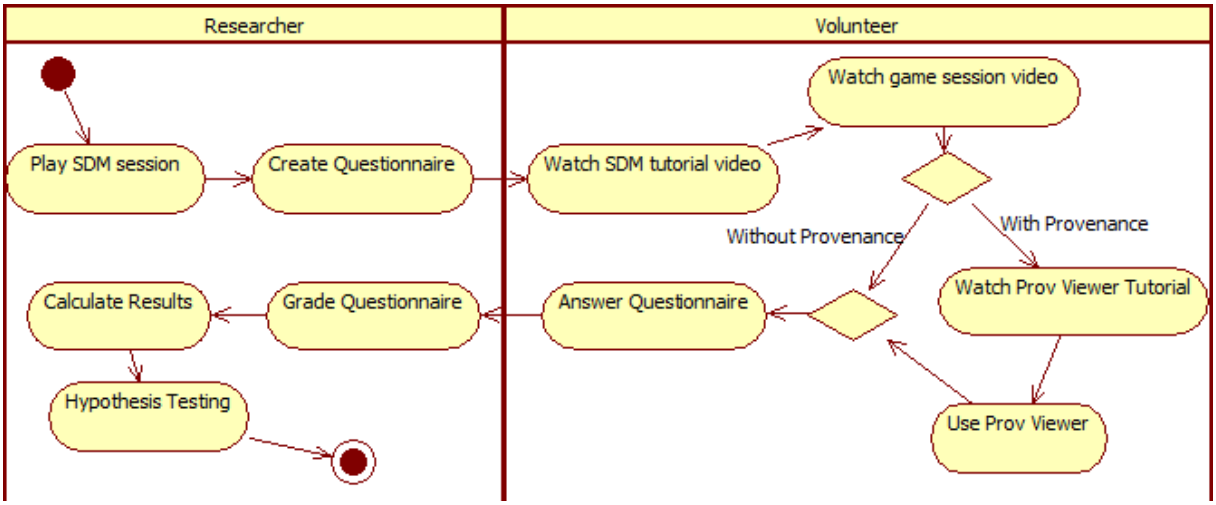

Figure 33: Experiment Execution activity diagram

The questionnaire was designed based on the video, consisting of ten questions. The first and the last questions are related to time measurements: the times when the volunteer started and finished the questionnaire. The second question is designed to identify the group of the volunteer: with provenance, which uses *Prov Viewer* while answering the questionnaire, or without provenance, which answers the questionnaire without using the tool. The other seven questions[9] are related to events that emerged during the game and have the same weight with values varying from 0 (wrong) to 1 (correct), depending on the answer provided. A value of 0.5 means the answer was partially correct, meaning that only one item was correctly identified. These questions explore different aspects from the game, and some questions require a deeper knowledge of the game.

The third question in the questionnaire asks one reason that made the employee Arden to quit the staff. The forth question is the same as the third, but for the employee Daniel. Their motives for quitting the staff were different. Arden left because of lack of payment (morale decreased due to lack of payment) while Daniel left due to overworking and lack of payment (morale decreased due to low stamina and lack of payment). Either answer was acceptable because we only asked one reason. The fifth question asks why Tornik had made no progress during a period of time. The sixth question asks why Daniel's productivity had a sudden drop from one day to another. The seventh question asks the most contributing factor that allowed finishing the software in time. The eighth question asks the two most contributing factors that caused financial problems after day eleven. The ninth, and last, question asks which employee was idle for a period of time.

[9] The questionnaire is available in Appendix D. Note that the questionnaire is in portuguese because we used brazilian volunteers.

The next stage is to run the experiment with volunteers. Before participating in the experiment, volunteers are required to read and sign a consent form. Then, volunteers watch a tutorial video from SDM, which explains details about the game interface, and read a written document summarizing key features. Subsequently, they watch the replay video and are divided in two groups: those that will use provenance and those that will not. After watching the replay video, the volunteers are handed the questionnaire. However, the group with provenance watches another tutorial video for the tool before receiving the questionnaire. This stage also has a time limit to avoid fatigue. All documents used at this stage are available in Appendix A, B, C, and D. Also, the game session is the same described in Chapter 5 and available in GEMS[10], along with the provenance graph. Lastly, we performed a statistical analysis over the results by means of hypothesis test in order to compare the obtained results from both methods (with and without provenance).

Another important factor for the design of the experiment concerns the definition of the significance level to be used during statistical analysis. For the experiments performed in this work we used a confidence interval of 95%, which translates to $\alpha = 0.05$ where α is the maximum probability of incorrect rejecting the null hypothesis (Type I error).

6.3 EXPERIMENT EXECUTION

The pilot was applied to an undergraduate class composed of 28 volunteers. The obtained results are described in Table 4 and Table 5, where μ represents the mean and σ the standard deviation. The duration values are expressed in minutes. However, this data was not used for the experiment or the statistical analysis due to the changes made after the pilot.

Table 4: Pilot Group with Provenance Results

	Results													μ	σ
Q3	0	0	1	0	0	0	0	1	1	0	0	1	0	0.3	0.48
Q4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Q5	0	1	1	0	0	0	1	1	1	0	1	1	0	0.54	0.52
Q6	0.5	1	1	0.5	0.5	0.5	0	0.5	1	0.5	1	0	0	0.54	0.37
Q7	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1	1	0.5	0.5	0.5	0	0.54	0.25
Q8	0	1	0.5	0.5	0.5	0.5	0.5	0	0	0.5	0.5	1	0.5	0.46	0.32
Duration	15	20	22	23	25	25	25	33	34	40	36	24	41	27.9	8.03

[10] The video and the *Prov Viewer* with the provenance graph used for the experiment are available at http://gems.ic.uff.br/ping/.

Table 5: Pilot Group without Provenance Results

	Results															μ	σ
Q3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.07	0.26
Q4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Q5	0	0	1	0	1	1	0	1	0	1	0	1	1	0	1	0.53	0.51
Q6	0	0	0	1	1	0	0	0	1	1	0	1	0	0	0	0.33	0.49
Q7	0	0.5	0.5	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.43	0.17
Q8	0	0.5	0.5	1	1	0	0	0.5	0.5	0	0.5	0	0.5	0.5	0	0.37	0.35
Duration	10	24	10	15	16	25	17	11	15	30	15	22	8	16	34	17.9	7.62

After the pilot and making the appropriate changes in the plan, we applied the experiment in two different undergrad classes, composed of 18 and 19 volunteers each. From those 37 volunteers, only 32 were able to finish the experiment in the allocated time, thus 5 partially answered questionnaires were discarded. Figure 34 illustrates the volunteer's characteristics, where the majority never heard about software engineering and was an under graduating student (SVAHNBERG *et al.*, 2008). After running the experiment on both classes, the questionnaires were analyzed and yielded the results described by Table 6 and Table 7.

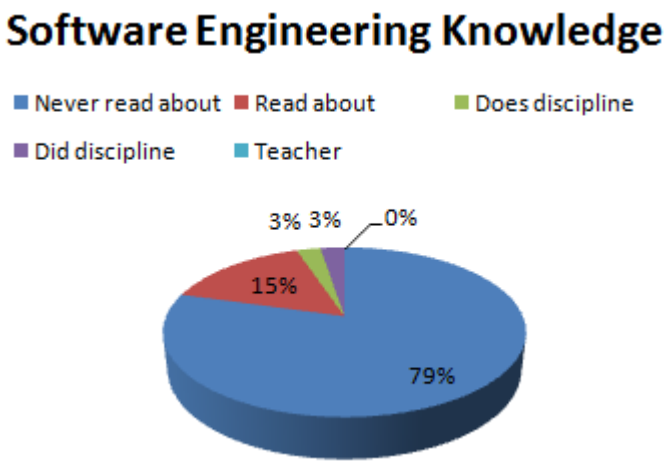

Figure 34: Volunteer's characterization results

Table 6: Group with Provenance Results

	Results																μ	σ
Q3	0	1	1	1	1	0	0	1	1	1	0	1	0	0	0	0	0.5	0.52
Q4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0.94	0.25
Q5	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0.19	0.4
Q6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q7	1	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1	0.37	0.5
Q8	0	0	0	0	0	0	0	0	0	1	0	0.5	0	0.5	0	0.5	0.16	0.3
Q9	1	1	1	1	1	1	1	1	1	0	1	0	1	0	1	1	0.81	0.4
Duration	25	18	19	21	18	19	21	21	28	21	28	29	26	27	20	30	23.19	4.25

Table 7: Group without Provenance Results

	Results																μ	σ
Q3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.06	0.25
Q4	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	0.87	0.34
Q5	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0.19	0.4
Q6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q7	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0.25	0.45
Q8	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0.5	0.5	0	0.09	0.2
Q9	1	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0.5	0.52
Duration	20	32	33	32	30	30	48	42	38	38	31	14	19	8	29	19	28.94	10.58

6.4 STATISTICAL ANALYSIS

A fundamental part of the statistical analysis of an experiment is the hypothesis test (WOHLIN *et al.*, 2000). In the hypothesis test, two hypotheses are proposed and used to validate the collected data. However, hypothesis testing involves two types of error: Type-I and Type-II. The Type-I error refers to the rejection of the null hypothesis even when it is true, while the Type-II error refers to the acceptance of the null hypothesis when it is false. These errors depend on the power of the test C, which is the probability of 1 - β that the test is true if H_0 is false and β is the probability of committing the error Type-II. Moreover, the hypothesis test can be parametric or non-parametric. Parametric tests have a greater power C, thus produces more accurate and precise estimates. However, parametric tests can only be used if samples follow a normal distribution. Nevertheless, non-parametric tests do not require normality and are recommended when samples are small (WOHLIN *et al.*, 2000).

The statistical analysis was performed with the intention of checking the obtained results and verifying if they have any significant difference. The main idea is to compare the results obtained from the questionnaire and the elapsed time of both groups. All tests were

done in the open source software R[11], which is commonly used for statistical analysis and graph construction, within the IDE *RStudio*[12].

6.4.1 NORMALITY TEST

On a normality test the null hypothesis H_0 states that the collected data follows a normal distribution. The alternative hypothesis, H_1, states that the collected data does not follow a normal distribution. Given this, a normality analysis from the obtained data decides between using parametric or non-parametric tests. Thus, we used the Shapiro-Wilk test (SHAPIRO; WILK, 1965) with the following hypotheses:

$$H_0: A\ sample\ x_1, x_2, \dots, x_n \quad have\ a\ normal\ distribution$$
$$H_1: A\ sample\ x_1, x_2, \dots, x_n \quad does\ not\ have\ a\ normal\ distribution$$

This test is executed in R by the command *shapiro.test(x)*, where *x* is the vector containing the data to be analyzed. It is provided as output the statistical value W[13] from the Shapiro-Wilk test and its *p-value*[14], as can be seen by Figure 35. The null hypothesis is rejected if *p-value* is lower than the significance level α, thus concluding that the data do not have a normal distribution.

```
> q3a <- c(0,1,1,1,1,0,0,1,1,1,0,1,0,0,0,0)
> shapiro.test(q3a)

        Shapiro-Wilk normality test

data:  q3a
W = 0.644, p-value = 4.34e-05
```

Figure 35: Example of R's output for Shapiro-Wilk test

The normality assumption was violated for all obtained results from the experiment because *p-value* < 0.01. It is possible to verify that *p-value* < α since α = 0.05 and *p-value* < 0.01, thus rejecting the null hypothesis. The results can be seen in Table 8. Note that for the group without provenance the duration *p-value* is greater than 0.05 (0.73). However, the group with provenance, where the *p-value* is 0.04337, which is smaller than α = 0.05. Null

[11] http://www.r-project.org/
[12] http://www.rstudio.com/
[13] The W statistic checks if the sample is from a normal distribution. Data normalization is shown by low values.
[14] *p-value* is the lowest level of significance at which the null hypothesis could be rejected for the given observations.

values in the tables come from constant observations, thus not being possible to test normality with Shapiro-Wilk, but indicating that the data does not follow a normal distribution.

Table 8: Normality Test Results with Outliers

	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Duration
With Prov	4.34e-05	4.553e-08	1.575e-08	Null	2.566e-05	1.213e-05	1.575e-06	0.04337
Without Prov	4.553e-08	3.408e-07	1.575e-06	Null	5.272e-06	1.33e-05	4.34e-05	0.7363

An important fact not observed until now was the presence of outliers. Outliers are data far from the norm for the population, typically with more than 1.5 interquartile range (Q3 – Q1) from other data. They can have detrimental effects on statistical analyses, increasing the variance error and reducing the power of statistical tests. However, not all outliers are illegitimate contaminants (BARNETT; LEWIS, 1994). The outliers detected in the sample are legitimate cases, since they are directly related to the correctness of the answers provided by volunteers. In any case, Table 9 illustrates the results for the normality test without outliers, which were removed following the 1.5 interquartile range definition.

Table 9: Normality Test Results without Outliers

	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Duration
With Prov	4.34e-05	Null	Null	Null	2.566e-05	3.481e-06	Null	0.04337
Without Prov	Null	Null	Null	Null	5.272e-06	Null	4.34e-05	0.7363

Therefore, non-parametric tests were used for statistical analysis. The test used to compare the means was Mann-Whitney, which is also known as Wilcoxon rank-sum[15] test. There are other non-parametric tests, such as Chi-2 and Kruskal-Wallis, however Mann-Whitney was chosen because it compares two means from two different samples against the same alternative hypothesis, which fits to our experiment design. The next section presents the results obtained from Mann-Whitney test to verify if the group results, with and without provenance, are equals.

6.4.2 COMPARISON OF MEANS

We adopted the following hypothesis in our tests:

$$H_0: \mu_{prov} = \mu_{replay}$$
$$H_1: \mu_{prov} \neq \mu_{replay}$$

[15] http://stat.ethz.ch/R-manual/R-patched/library/stats/html/wilcox.test.html

The mean is calculated for each question from the questionnaire and for the duration that each volunteer took to finish it. Table 10 illustrates the mean the standard deviation of each question for both methods, based on the data presented in Table 6 and Table 7.

Table 10: Mean and Standard Deviation for each question

		Q3	Q4	Q5	Q6	Q7	Q8	Q9	Duration
With Prov	**Mean**	0.5	0.9375	0.1875	0	0.375	0.1562	0.8125	23.1875
	Standard Deviation	0.5164	0.25	0.4031	0	0.5	0.3010	0.4031	4.2461
Without Prov	**Mean**	0.0625	0.875	0.1875	0	0.25	0.0938	0.5	28.9375
	Standard Deviation	0.25	0.3416	0.4031	0	0.4472	0.2015	0.5162	10.5797

The *boxplots* shown in Figure 36 aims at summarizing the distributions of both with and without provenance methods. In these graphs, the boxes represent part of the central distribution, which contains 50% of data. Thus, the data scattering is proportional with the box's height. The median is represented by a black line inside the box. This way, 25% of data is between the box's edges and the median. The median location indicates if the distributions are symmetrical in the experiments. Lastly, circles indicate outliers.

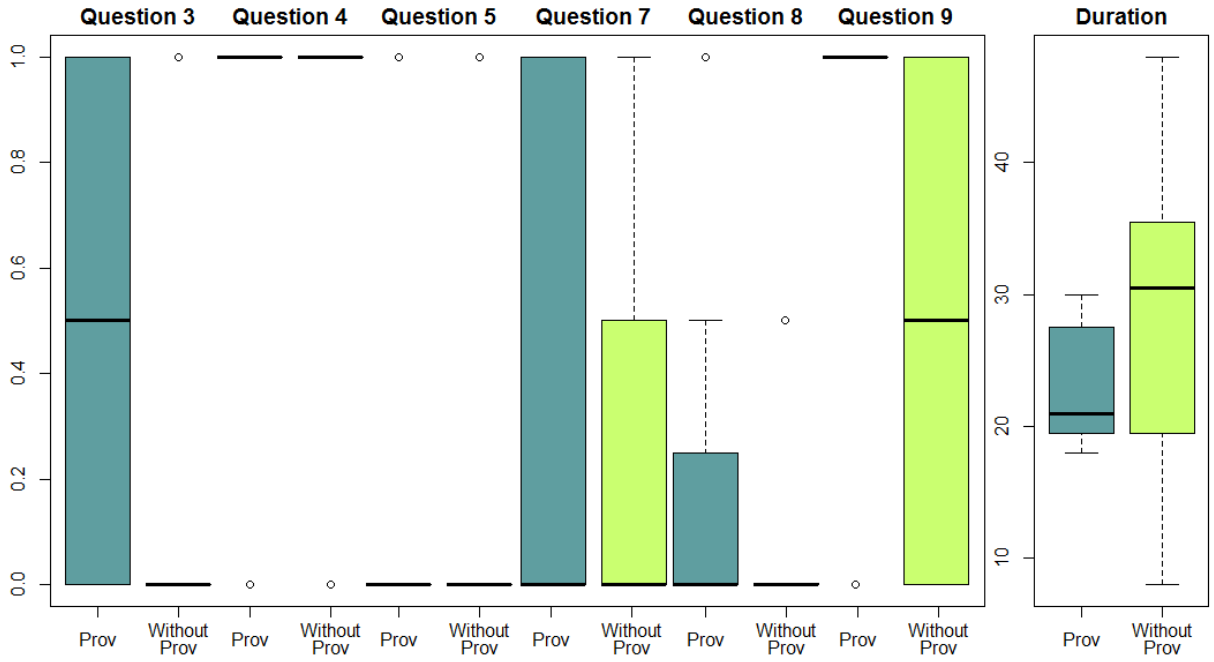

Figure 36: Boxplots from the experiment

It is possible to assert that there is a difference in means if the null hypothesis is rejected. The Mann-Whitney test is performed in R by the command *wilcox.test(x, y, conf.int = T)*, where *x* and *y* are vectors to be tested and *conf.int* is used to display the confidence interval. As default, the *wilcox.test* paired attribute is set to false, representing the Mann-

Whitney test. Figure 37 illustrates an example of the output from this command in R with the default α value, which is 0.05, while Table 11 illustrates all obtained results.

```
> wilcox.test(q3a ,q3b ,conf.int =T)

        Wilcoxon rank sum test with continuity correction

data:  q3a and q3b
W = 184, p-value = 0.007259
alternative hypothesis: true location shift is not equal to 0
95 percent confidence interval:
 3.136373e-05 1.000000e+00
sample estimates:
difference in location
          4.594792e-05
```

Figure 37: R's output for Mann-Whitney test

Table 11: Results obtained from the Mann-Whitney test

α = 0.05	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Duration
p-value	0.007259	0.5757	1	Null	0.467	0.6371	0.07049	0.03595
CI	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001	< 0.001

The null hypothesis is not rejected if *p-value* is greater than significance level α. In other words, there is not enough evidence to assert a difference between results. When the null hypothesis is rejected (*p-value* < α), it is necessary to identify which method is superior by analyzing the confidence interval *CI*. If *CI* – α < 0, then $\mu_{prov} > \mu_{replay}$. Otherwise $\mu_{prov} < \mu_{replay}$. By analyzing the *p-values* from Table 11, the usage of provenance analysis provided better results in question 3 and in the time required to finish the questionnaire (duration), while there is not enough evidence to assert difference between results for the other questions (*p-value* > α). Even though both questions 3 and 4 asked the same thing, one reason that led the employee to quit the staff, only one volunteer that answered the questionnaire without provenance identified the lack of payment as the reason for question 3.

By analyzing the *boxplots* in Figure 36, it is possible to infer that question 3 yielded better results by using provenance while questions 4 and 5 had equal results. Meanwhile, questions 7 and 8 results were similar but with varying scattering. Even though results are matching with Mann-Whitney test data, question 9 has a different behavior due to the small difference from *p-value* to α (*p-value* = 0.07 against α = 0.05). By analyzing the *boxplot* for question 9, the results for using provenance are greater than without provenance. While without provenance's data is scattered around the maximum and minimum values with the

median at the middle, the provenance's median is located at the maximum value. Lastly, as shown by the Mann-Whitney test, using provenance for analysis provides faster answers than analyzing the game session's replay. This is clearly seen by comparing the medians between both methods and the box's scattering (height) position. The *boxplot* for question 6 was discarded because both methods had equal values and were all zero (without outliers).

6.5 THREATS TO VALIDITY

Despite the care in reducing the threats to the validity of the experiment, there are factors that can influence the results. In relation to internal validity, the selection for both groups (with provenance and without provenance) can affect the results because of the natural variation in human performance. Furthermore, the experiment was executed with volunteers, which generally are more motivated for executing tasks. Anyone from the class could choose to be dismissed from the experiment and be released earlier. Lastly, the experiment was the first contact of the volunteers with both the game mechanics (by watching the video) and the tool. Thus, the lack of experience can affect the results, even when minimized by the usage of tutorials. For external validity, to level the experience of volunteers, they were from two different classes of the same discipline (Introduction to computer programming), which occurs in the first period of undergraduate course in Computer Science at *Universidade Federal Fluminense*.

Regarding construct validity, the questionnaires were composed of several questions to reduce threats related to a lack of knowledge from the game, thus exploring different aspects from it. Another risk is related to people being afraid of being evaluated, thus trying to "look better" by lying. This is the case of how long they took to finish answering the questionnaire. To minimize this, we had a strict timetable for each activity, stating the exact time they began answering the questionnaire and verifying the time they finished and delivered the questionnaire.

A threat related to conclusion validity is the reliability of measures. This is dependent on factors like question wording, which may allow for different interpretations, and the graph layout. To minimize the threat, we answered any doubts voiced by volunteers related to the questions in the questionnaire or regarding the tool (*Prov Viewer*).

6.6 FINAL CONSIDERATIONS

This chapter presented the evaluation of the adoption of provenance to analyze a game flux. This evaluation was performed using statistical analysis on the values obtained from the

experiments. The results demonstrate that results analyzed based with provenance were equal or greater than watching a replay. Furthermore, analyzing the game flux with provenance is faster than only watching a replay of the game session.

In relation to correctly identify the causes of the events in the game, using provenance provided better statistical results in at least one case (question 3, related to lack of payment), and slightly better results in another (question 9, related to identifying the idle employee). The other cases were not statistically different with the current sample size.

CHAPTER 7 – CONCLUSION

7.1 CONTRIBUTIONS

This work introduced a new approach for gameplay data logging and visualization, entitled *Provenance in Games*. This approach collects gameplay information from executed actions and events records related information, including the characters involved, those that were affected by the action or event, and the generated influences. It also records game states for each entity (characters and objects) and the influences that changed their states. After the completion of the gaming session, the collected information, denominated as *game flux log*, is exported to an external provenance graph visualization tool: *Prov Viewer*. The *game flux log* contains provenance information from the gaming session and is used by *Prov Viewer* to plot the game's provenance graph.

The provenance graph allows post game analysis to discover issues that contributed to specific game fluxes and results achieved throughout the game session. This analysis can be used to improve understanding of the game flux and to identify actions that influenced the outcome, aiding the player to understand why they happened the way they did. It can also be used to analyze the game story development, how it was generated, and which events affected it.

Prov Viewer uses graphic features to distinguish information for faster comprehension of the events. These features affect the displayed graph by transforming vertices and edges, changing their shapes and color according to the information type. Another important feature present is the information filter, which omit displayed information that is not relevant for the analysis. This filtering is important for analysis because it reduces the amount of displayed information to those that are the focus of interest of the user. This allows for faster identification of the influences in the game, which is possible due the way the provenance graph is structured.

Both precision and agility were evaluated by an experiment, where volunteers watched a game session and answered a questionnaire containing specific questions about certain events that occurred in the game. Half the volunteers in the experiment had access to the provenance graph while the other half had only access to the replay. As a result, we could conclude that in the experiment context the usage of provenance graph for analysis provided faster and more precise answers when determining the reasons of outcomes in a game, in comparison with watching a replay of the game session.

While the main application of provenance in this work is over a serious game and is used to assist players in understanding how events affected the story, we believe that the concepts discussed here are applicable to other kinds of games and useful to support advanced forms of analysis. These concepts may be useful for gameplay balancing and design, data mining of behaviors, storytelling, gameplay metrics, and aiding developers by detecting gameplay issues.

Table 12 extends the comparative chart among approaches in Section 2.6 of Chapter 2 by also comparing them with the proposed approach of this work, *Provenance in Games*.

Table 12: Comparative chart among approaches

Features	GVM (JOSLIN *et al.*, 2007)	TRUE (KIM *et al.*, 2008)	Playtracer (ANDERSEN *et al.*, 2010)	Play-Graph (WALLNER, 2013)	Provenance in Games
Graph			√	√	√
Graphic	√	√			
State Machine		?	√	√	√
Data Logging	√	√	?	?	√
Event Context		√			√
Player Behavior	?	√	√	√	√
Actions		√		√	√
Statistical Data Mining	√	√	√	√	√
Developer-Oriented	√	√	√	√	√
Player-Oriented					√
Cause-Effect					√

7.2 LIMITATIONS

One limitation of the *Prov Viewer* prototype is related to scalability, regarding processing performance. Algorithms in *Prov Viewer* are not optimized for manipulating a graph with thousands vertices and edges, although we developed a simple level of detail method. Thus, its performance may degrade when dealing with such graph sizes. A second

limitation is the need of tinkering with the source code in order to make *Prov Viewer* compatible with other games or provenance applications.

Another limitation is related to *Prov Viewer's* input file format. Currently, *Prov Viewer* is not compatible with other provenance applications due to the input file format, which is a text-based structure exported by SDM. Nevertheless, it can be adapted for known formats, such as JSON or XML, by modifying how *Prov Viewer* reads a file. We have plans to modify the structure to use some semi-structured format such as JSON or XML for greater compatibility with other applications. Thus, allowing *Prov Viewer* to be more accessible by other applications due to the usage of a well known semi-structured format.

7.3 FUTURE WORK

After developing the conceptual framework for data logging and analysis, along with *Prov Viewer*, it is possible to describe new research possibilities for the proposed approach. The following paragraphs describe possible researches and improvements in the *Provenance in Games* conceptual framework and in the *Prov Viewer* provenance graph visualization tool.

A possible improvement in the visualization tool is related with accessibility, by making *Prov Viewer* less context sensitive, allowing the user to customize filters (edge filters and attribute status visualization) without the need of hard coding it in the application. For example, we could allow the user to provide a configuration file that specifies the type of each filter. Thus, this would make *Prov Viewer* compatible with other games or provenance applications without the need of tinkering with the source code.

One hypothesis for greatly improving accessibility for *Prov Viewer* would be in creating an extension for existing game engines (i.e., Unity3D). This extension, when enabled, would automatically capture gameplay information when executing events by using the *Provenance in Games* conceptual framework. Thus, it would be possible to generate a *game flux log* using a known format (XML) without the need of changing the game's source code.

Another possible research option for *Prov Viewer* is related to inferences. Currently, it does not automatically make inferences to the user, but let the user or developers decide what needs to be inferred. This might cause visualizations problems at a first glance due to the size of the graphs at their full extension, overwhelming the user. However, we provide the necessary tools to create inference rules, like filters and collapses (both for vertices and edges). Studies in this area can be made in order to identify information that can be omitted

from the user without affecting the overall analysis, while at the same time not being context dependable. Thus it would provide generic inferences rules that could be used in any game.

Another area to be worked upon is related to the graph visualization scalability. Depending on the game style, a game session might take several hours to complete, or even days in case of RPGs. This makes the size of the provenance graph to be overwhelming to the user, even when removing unnecessary information during the generation of the *game flux log* to reduce the log size. One way to avoid such situations is to show the provenance graph with different levels of details, applying some filters instead of displaying the graph's full extension. For example, before showing the graph to the user, it is possible to use collapses to reduce the graph's size. Combats can be identified and collapsed into a single vertex for each instance. Places visited in the game can also be collapsed into a single vertex, containing all interactions made in that location, even combats. It is also possible to have collapses inside collapses. In this case, a collapsed combat inside a collapsed area visited by the player may contain other actions aside from the combat, such as interactions with the ambient. This gives an impression of a map from the player's journey, showing vertices for each location visited by the player, while allowing the player to expand only the situations he desires to analyze. It is similar to *google maps*, where it shows the entire world and allows the user to zoom into specific locations. However in this case, it shows instances of the journey taken by the player.

It is also possible to go beyond that by using automatic inferences algorithms. Instead of collapsing all combats and locations, inference algorithms could decide which combats or locations were not relevant to the story, or had no noticeable impact in the player's journey, while keeping important events visible to the player. This is possible because provenance is analyzed from the present to the past. This way, combats' outcomes are known and can be used to decide if they are relevant or not. If the player was victorious with minor challenge, did not suffer severe wounds, or barely used any resources at his disposal, then the entire combat can be simplified into just one vertex, representing the combat with the enemy. However, if the combat was challenging or the player lost, it is interesting to display all actions in it for analysis, allowing the player to identify important facts that influenced the combat outcome.

On this work, we only used the core structure from the PROV model (agent, activity, entity). However, there are other interesting provenance elements described at the PROV model that could be studied and incorporated to further distinguish and classify the game's provenance information. Some of these elements are: bundles, collections, and more categories of agents (*i.e. software agent* and *organization*).

Although the proposed approach was used for assisting players to understand game events, we believe that it can also be used to open new researches in the field of behavior patterns data mining, gameplay design, detection of gameplay issues, and adjustment of gameplay features. We also believe that the proposed approach can be used for game balancing in real time through automatic provenance analysis and data mining.

BIBLIOGRAPHY

ABT, Clark C. *Serious Games*. 1. ed. Abt Books: University Press of America, 1987.

AMBINDER, Mike. Valve's approach to playtesting: The application of empiricism. *Game Developer Conference (GDC)*, 2009.

ANDERSEN, Erik; LIU, Yun-En; APTER, Ethan; BOUCHER-GENESSE, François; POPOVIĆ, Zoran. Gameplay analysis through state projection. *Foundations of Digital Games (FDG)*, p. 1–8, 2010.

BAEZA-YATES, Ricardo A.; RIBEIRO-NETO, Berthier. *Modern Information Retrieval*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

BAKER, Alex; NAVARRO, Emily; VAN DER HOEK, André. Problems and Programmers: An Educational Software Engineering Card Game. *International Conference on Software Engineering (ICSE)*, p. 614–621, 2003.

BARNETT, Vic; LEWIS, Toby. *Outliers in Statistical Data*. Chichester; New York: Wiley, 1994.

BIVAR, Bárbara; SANTOS, Lucas; KOHWALTER, Troy; MARINHO, Anderson; MATTOSO, Marta; BRAGANHOLO, Vanessa. Uma Comparação entre os Modelos de Proveniência OPM e PROV. *Brazilian e-Science workshop*, 2013.

BOSE, Rajendra; FOSTER, Ian; MOREAU, Luc. Report on the International Provenance and Annotation Workshop: (IPAW'06) 3-5 May 2006, Chicago. *ACM's Special Interest Group on Management Of Data (SIGMOD)*, v. 35, n. 3, p. 51–53, 2006.

BRISTOL, Edgar H. Pattern recognition: An alternative to parameter identification in adaptive control. *Automatica*, v. 13, n. 2, p. 197–202, 1977.

CHENEY, James. *Semantics of the PROV Data Model*. Available: <http://www.w3.org/TR/2013/WD-prov-sem-20130312/>. Accessed: 26 mar. 2013.

CHIALVO, Dante R.; BAK, Per. Learning From Mistakes. *Neuroscience*, v. 90, n. 4, p. 1137–1148, 1999.

CIOS, K.J.; PEDRYCZ, W.; SWINIARSK, R.M. Data Mining Methods for Knowledge Discovery. *IEEE Transactions on Neural Networks*, v. 9, n. 6, p. 1533–1534, 1998.

CLARK, George. The organization of behavior: A neuropsychological theory. *The Journal of Comparative Neurology*, v. 93, n. 3, p. 459–460, 1950.

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

DANKOFF, Jonathan. Game telemetry with playtest DNA on Assassin's Creed. *Ubisoft Montreal*, 2011. Available: <http://engineroom.ubi.com/game-telemetry-with-playtest-dna-on-assassins-creed/>.

DAVIDSON, Susan B.; FREIRE, Juliana. Provenance and scientific workflows: challenges and opportunities. *ACM's Special Interest Group on Management Of Data (SIGMOD)*, p. 1345–1350, 2008.

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.

DIEHL, Stephan. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

DIXIT, Priyesh; YOUNGBLOOD, Michael. Understanding playtest data through visual data mining in interactive 3D environments. *12th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia and Serious Games (CGAMES)*, p. 34–42, 2008.

DRACHEN, Anders; CANOSSA, Alessandro. Analyzing spatial user behavior in computer games using geographic information systems. *MindTrek Conference: Everyday Life in the Ubiquitous Era*, p. 182–189, 2009.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, v. 17, n. 3, p. 37, 1996.

FREIRE, J.; KOOP, D.; SANTOS, E.; SILVA, C.T. Provenance for Computational Tasks: A Survey. *Computing in Science Engineering*, v. 10, n. 3, p. 11–21, 2008.

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

GARIJO, Daniel; ECKERT, Kai; MILES, Simon; TRIM, Craig M.; PANZER, Michael. *Dublin Core to PROV Mapping*. Available: <http://www.w3.org/TR/2013/WD-prov-dc-20130312/>. Accessed: 26 mar. 2013.

GIL, Yolanda; CHENEY, James; GROTH, Paul; HARTIG, Olaf; MILES, Simon; MOREAU, Luc; SILVA, Paulo. *W3C Provenance Incubator Group*. Available: <http://www.w3.org/2005/Incubator/prov/wiki/Main_Page>. Accessed: 22 mar. 2013.

GIL, Yolanda; DEELMAN, Ewa; ELLISMAN, Mark; FAHRINGER, Thomas; FOX, Geoffrey; GANNON, Dennis; GOBLE, Carole; LIVNY, Miron; MOREAU, Luc; MYERS, Jim. Examining the Challenges of Scientific Workflows. *Computer*, v. 40, n. 12, p. 24–32, 2007.

GIL, Yolanda; MILES, Simon. *PROV Model Primer*. Available: <http://www.w3.org/TR/prov-primer/>. Accessed: 21 mar. 2013.

GOODMAN, Leo A. Snowball Sampling. *The Annals of Mathematical Statistics*, v. 32, n. 1, p. 148–170, 1961.

GROTH, Paul; LEBO, Timothy; MOREAU, Luc; SOILAND-REYES, Stian; MISSIER, Paolo; SAHOO, Satya. *ProvImplementations*. Available: <http://www.w3.org/2011/prov/wiki/ProvImplementations>. Accessed: 26 mar. 2013.

GROTH, Paul; MOREAU, Luc. *PROV-Overview*. Available: <http://www.w3.org/TR/prov-overview/>. Accessed: 26 mar. 2013.

HAN, Jiawei; KAMBER, Micheline. *Data Mining: Concepts and Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

HIGGINS, T. *Unity - 3D Game Engine*. Available: <http://unity3d.com/>. Accessed: 5 maio 2011.

HOOBLER, Nate; HUMPHREYS, Greg; AGRAWALA, Maneesh. Visualizing Competitive Behaviors in Multi-User Virtual Environments. *Visualization (VIS)*, p. 163–170, 2004.

HUA, Hook; TILMES, Curt; ZEDNIK, Stephan; MOREAU, Luc. *PROV-XML: The PROV XML Schema*. Available: <http://www.w3.org/TR/prov-xml/>. Accessed: 26 mar. 2013.

JOSHUA O'MADADHAIN; DANYEL FISHER; TOM NELSON. *JUNG: Java Universal Network/Graph Framework*. Available: <http://jung.sourceforge.net/>.

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, p. 6, 2007.

KIM, Jun H.; GUNN, Daniel V.; SCHUH, Eric; PHILLIPS, Bruce; PAGULAYAN, Randy J.; WIXON, Dennis. Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. *Human Factors in Computing Systems (CHI)*, p. 443–452, 2008.

KOHWALTER, Troy; CLUA, Esteban; MURTA, Leonardo. Provenance in Games. *Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, 2012.

KOHWALTER, Troy; CLUA, Esteban; MURTA, Leonardo. SDM – An Educational Game for Software Engineering. *Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, p. 222–231, 2011.

LEBO, Timothy; SAHOO, Satya; MCGUINESS, Deborah. *PROV-O: The PROV Ontology*. Available: <http://www.w3.org/TR/prov-o/>. Accessed: 21 mar. 2013.

LIU, Yun-En; ANDERSEN, Erik; SNIDER, Richard; COOPER, Seth; POPOVIĆ, Zoran. Feature-based projections for effective playtrace analysis. *Foundations of Digital Games (FDG)*, p. 69–76, 2011.

MILES, Simon; HEASLEY, Jim; SZALAY, Alex; MOREAU, Luc; GROTH, Paul. *Provenance Challenge WIKI*. Available: <http://twiki.ipaw.info/bin/view/Challenge/>. Accessed: 26 mar. 2013.

MISSIER, Paolo; MOREAU, Luc; CHENEY, James; LEBO, Timothy; SOILAND-REYES, Stian; NIES, Tom De; COPPENS, Sam. *PROV Dictionary*. Available: <http://www.w3.org/TR/2013/WD-prov-dictionary-20130312/>. Accessed: 26 mar. 2013.

MOREAU, Luc. *OPM Toolbox*. Available: <http://openprovenance.org/toolbox.html>. Accessed: 2 abr. 2013a.

MOREAU, Luc. *OPM4J: The Open Provenance Model Java Library*. Available: <http://openprovenance.org/java/site/1_1_8/apidocs/org/openprovenance/model/package-summary.html>. Accessed: 2 abr. 2013b.

MOREAU, Luc; CLIFFORD, Ben; FREIRE, Juliana; FUTRELLE, Joe; GIL, Yolanda; GROTH, Paul; KWASNIKOWSKA, Natalia; MILES, Simon; MISSIER, Paolo; MYERS, Jim; PLALE, Beth; SIMMHAN, Yogesh; STEPHAN, Eric; DEN BUSSCHE, Jan Van. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, v. 27, n. 6, p. 743–756, 2007.

MOREAU, Luc; DING, Li; FUTRELLE, Joe; GARIJO, Daniel; GROTH, Paul; JEWELL, Mike; MILES, Simon; MISSIER, Paolo; PAN, Jeff; ZHAO, Jun. *Open Provenance Model (OPM) OWL Specification*. Available: <http://openprovenance.org/model/opmo>. Accessed: 2 abr. 2013.

MOREAU, Luc; FOSTER, Ian; FREIRE, Juliana; FREW, James; GROTH, Paul; MCGUINESS, Deborah. *IPAW*. Available: <http://www.ipaw.info/>. Accessed: 2 abr. 2013.

MOREAU, Luc; GROTH, Paul; CLIFFORD, Ben; MILES, Simon. *Open Provenance Model (OPM) XML Schema Specification*. Available: <http://openprovenance.org/model/opmx>. Accessed: 2 abr. 2013.

MOREAU, Luc; LEBO, Timothy. *Linking Across Provenance Bundles*. Available: <http://www.w3.org/TR/2013/WD-prov-links-20130312/>. Accessed: 26 mar. 2013.

MOREAU, Luc; MISSIER, Paolo. *PROV-DM: The PROV Data Model*. Available: <http://www.w3.org/TR/prov-dm/>. Accessed: 21 mar. 2013a.

MOREAU, Luc; MISSIER, Paolo. *PROV-N: The Provenance Notation*. Available: <http://www.w3.org/TR/prov-n/>. Accessed: 21 mar. 2013b.

MORET, Bernard. Decision Trees and Diagrams. *ACM Computing Surveys (CSUR)*, v. 14, n. 4, p. 593–623, 1982.

NASCIMENTO, Giancarlo; MOURÃO, Pedro; RUFF, Christian; TENÓRIO, Alisson; CLUA, Esteban; SANTOS, Venétia; ZAMBERLAN, Cristina. A real-time simulator for ergonomics and displacement evaluations. *Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, 2010.

NAVARRO, Emily Oh; VAN DER HOEK, André. SimSE: an educational simulation game for teaching the Software engineering process. *ACM Special Interest Group on Computer Science Education (SIGCSE)*, v. 36, n. 3, p. 233–233, 2004.

NIES, Tom De; CHENEY, James; MISSIER, Paolo; MOREAU, Luc. *Constraints of the PROV Data Model*. Available: <http://www.w3.org/TR/prov-constraints/>. Accessed: 21 mar. 2013.

PASSOS, Erick Baptista; JOSELLI, Mark; ZAMITH, Marcelo; CLUA, Esteban Walter Gonzalez; MONTENEGRO, Anselmo; CONCI, Aura; FEIJO, Bruno. A bidimensional data structure and spatial optimization for supermassive crowd simulation on GPU. *Computers in Entertainment (CIE)*, v. 7, n. 4, p. 60, 2009.

PREMIS WORKING GROUP. *Data Dictionary for Preservation Metadata*. Final report. OCLC Online Computer Library Center & Research Libraries Group: Implementation Strategies (PREMIS), 2005.

PRENSKY, Marc. Fun, Play and Games: What Makes Games Engaging. *Digital Game-Based Learning*, p. 1–31, 2001.

ROMERO, Ramon. Successful instrumentation: Tracking attitudes and behaviors to improve games. *Game Developer Conference (GDC)*, 2008.

SCHULTZ, Warren. *AAA Game*. Available: <http://gameindustry.about.com/od/glossary/g/Aaa-Game.htm>. Accessed: 3 jul. 2013.

SHAPIRO, S. S.; WILK, M. B. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika*, v. 52, n. 3/4, p. 591, 1965.

SIMMHAN, Yogesh L.; PLALE, Beth; GANNON, Dennis. A survey of data provenance in e-science. *ACM's Special Interest Group on Management Of Data (SIGMOD)*, v. 34, n. 3, p. 31–36, set. 2005.

SVAHNBERG, Mikael; AURUM, Aybüke; WOHLIN, Claes. Using students as subjects - an empirical evaluation. *Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM)*, p. 288–290, 2008. Accessed: 21 ago. 2013.

THOMPSON, Clive. Halo 3: How Microsoft Labs Invented a New Science of Play. *Wired Magazine*, v. 15, n. 9, 2007.

THOMPSON, Jim; BERBANK-GREEN, Barnaby; CUSWORTH, Nic. *Game Design: Principles, Practice, and Techniques - The Ultimate Guide for the Aspiring Game Designer*. 1. ed. United States: Wiley, 2007.

WALLNER, Günter. Play-Graph: A Methodology and Visualization Approach for the Analysis of Gameplay Data. *Foundations of Digital Games (FDG)*, p. 253–260, 2013.

WALLNER, Günter; KRIGLSTEIN, Simone. A spatiotemporal visualization approach for the analysis of gameplay data. *Human Factors in Computing Systems (CHI)*, p. 1115–1124, 2012.

WEITZNER, Daniel J.; ABELSON, Harold; BERNERS-LEE, Tim; FEIGENBAUM, Joan; HENDLER, James; SUSSMAN, Gerald Jay. Information accountability. *Communications of the ACM*, v. 51, n. 6, p. 82–87, 2008.

WITTEN, Ian; FRANK, Eibe. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

WOHLIN, Claes; RUNESON, Per; HÖST, Martin; OHLSSON, Magnus C.; REGNELL, Bjöorn; WESSLÉN, Anders. *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

ZHAO, Jun. *Open Provenance Model Vocabulary Specification*. Available: <http://open-biomed.sourceforge.net/opmv/ns.html>. Accessed: 2 abr. 2013.

ZOELLER, Georg. Development telemetry in video games projects. *Game Developer Conference (GDC)*, 2010.

APPENDIX A – CONSENT FORM

Estudo

Este estudo visa avaliar o quanto as técnicas de Proveniência em Jogos, apresentadas através de um cenário de jogo do SDM, são beneficentes para o aprendizado, tanto para um aluno experiente quanto para um aluno novo no assunto.

Idade

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por *Troy Costa Kohwalter* da Universidade Federal Fluminense.

Procedimento

Este estudo acontecerá em uma única sessão, que incluirá a apresentação de um vídeo do jogo SDM, documentos auxiliares para o entendimento do procedimento, um questionário, e em alguns casos a utilização da ferramenta *Proof Viewer*. Eu entendo que, uma vez que o experimento tenha terminado, os trabalhos que desenvolvi serão estudados visando entender a eficácia do modelo proposto.

Confidencialidade

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e apresentado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação relacionada à minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas de ensino para a Engenharia de Software.

Pesquisador responsável
Troy Costa Kohwalter
Instituto de Computação – Universidade Federal Fluminense (UFF)
Professores responsáveis (Orientadores)
Prof Leonardo Gresta Paulino Murta
Instituto de Computação – Universidade Federal Fluminense (UFF)
Prof Esteban W. Gonzalez Clua
Instituto de Computação – Universidade Federal Fluminense (UFF)

Nome:_____
Assinatura: _____
Data:___/___/___

Este formulário está dividido Formulário de Consentimento, Questionário de Caracterização e um Questionário de Conteúdo.

Desde já, agradecemos a sua disponibilidade.

APPENDIX B – CHARACTERISATION QUESTIONNAIRE

1) Formação Acadêmica

   ( ) Doutorado

   ( ) Doutorando

   ( ) Mestrado

   ( ) Mestrando

   ( ) Graduação

   ( ) Graduando

   Ano de ingresso: _____ Ano de conclusão (ou previsão de conclusão): _____

2) Formação Geral

   a. Qual é sua experiência em Engenharia de Software? (marque aqueles itens que melhor se aplicam)

      ( ) Nunca aprendi Engenharia de Software.

      ( ) Já li material sobre Engenharia de Software.

      ( ) Estou fazendo uma disciplina sobre Engenharia de Software.

      ( ) Já fiz uma disciplina de Engenharia de Software.

      ( ) Dou aula de Engenharia de Software.

APPENDIX C – PILOT EXPERIMENT QUESTIONNAIRE

Questionário de Avaliação do Conteúdo

1) Hora de inicio: _____
2) Utilizou o *Proof Viewer*?
   ( ) Sim.
   ( ) Não.
3) Considerando que nos dias três e quatro, o funcionário Urias exerceu a mesma tarefa (Elicitação sem protótipo), por que seu desempenho foi quase um terço (1/3) no dia quatro em comparação ao dia três?

   _____

   _____
4) Descreva os motivos que levaram a funcionária Emmy pedir demissão no dia 15.

   _____

   _____
5) Identifique a semana com o maior índice de produtividade. Aponte os fatores que levaram essa conclusão.

   _____

   _____
6) Identifique a semana com menor índice de produtividade. Aponte os fatores que levaram essa conclusão.

   _____

   _____
7) Identifique os fatores que levaram a demissão de diversos funcionários durante a quinta e sexta semanas (dias 26 a 34).

   _____

   _____
8) Identifique os fatores mais contribuintes que levaram a falta de Creditos apresentada na quarta semana (dias 20 a 26).

   _____

   _____
9) Hora de término: _____

Questionário de Avaliação do Conteúdo

Gabarito

(Resposta similar ou com mesmo sentido/significado é considerado correto)

1) Hora de inicio: _____
2) Utilizou o *Proof Viewer*?
   ( ) Sim.
   ( ) Não.
3) Considerando que nos dias três e quatro, o funcionário Urias exerceu a mesma tarefa (Elicitação sem protótipo), por que seu desempenho foi quase um terço (1/3) no dia quatro em comparação ao dia três?
   Por causa da influencia negativa da Edda
4) Descreva os motivos que levaram a funcionária Emmy pedir demissão no dia 15.
   Horas extras decrementaram a estamina e consequentemente o moral
5) Identifique a semana com o maior índice de produtividade. Aponte os fatores que levaram essa conclusão.
   Semana 3 (dias 14-20). Alteração da carga horária de 8 para 16 horas diárias
6) Identifique a semana com menor índice de produtividade. Aponte os fatores que levaram essa conclusão.
   Semana 5 (dias 28-34). Estamina e Moral baixos. Iniciou a semana com metade da equipe por causa de pedidos de demissão
7) Identifique os fatores que levaram a demissão de diversos funcionários durante a quinta e sexta semanas (dias 26 a 34).
   Moral baixa por causa de horas extras e falta de pagamentos nos dias 23 a 28
8) Identifique os fatores mais contribuintes que levaram a falta de Creditos apresentada na quarta semana (dias 20 a 26).
   Horas extras dobraram as despesas e as contratações iniciais
9) Hora de término: _____

APPENDIX D – EXPERIMENT QUESTIONNAIRE

Questionário de Avaliação do Conteúdo

1) Hora de inicio: _____
2) Utilizou o *Proof Viewer*?
   ( ) Sim.
   ( ) Não.
3) Qual foi o motivo responsável pela redução do moral do funcionário *Arden* que consequentemente levou a seu pedido de demissão?

   _____

4) Qual foi o motivo responsável pela redução do moral do funcionário *Daniel* que consequentemente levou a seu pedido de demissão?

   _____

5) Por que nos dias 9, 10, e 11 o funcionário *Tornik* não obteve progresso na sua função de *elicitação* (não teve aumento nos requisitos de cliente)?

   _____

   _____

6) Por que nos dias 10 e 11 o rendimento de Daniel na sua função de *especificação* teve uma discrepância muito grande (342 *validation* vs 34 *validation*)?

   _____

   _____

7) Mesmo entregando o projeto com *bugs* não encontrados/corrigidos, qual foi o maior fator contribuinte que permitiu entregar o projeto a tempo?

   _____

   _____

8) Identifique os dois fatores mais contribuintes que levaram a falta de Creditos apresentada a partir do dia 11.

   _____

   _____

9) Um funcionário ficou sem nenhuma tarefa durante quatro dias. Quem foi esse funcionário?

   _____

   _____

10) Hora de término: _____

    Se tiver alguma sugestão, escreva no verso. Obrigado.

Experimento: Questionário de Avaliação do Conteúdo

Gabarito

(Resposta similar ou com mesmo sentido/significado é considerado correto)

1) Hora de inicio: _____
2) Utilizou o *Proof Viewer*?
   ( ) Sim.
   ( ) Não.
3) Qual foi o motivo responsável pela redução do moral do funcionário *Arden* que consequentemente levou a seu pedido de demissão?

   Falta de pagamentos

4) Qual foi o motivo responsável pela redução do moral do funcionário *Daniel* que consequentemente levou a seu pedido de demissão?

   Exaustão por horas extras ou falta de pagamentos (foi pedido apenas um motivo)

5) Por que nos dias 9, 10, e 11 o funcionário *Tornik* não obteve progresso na sua função de *elicitação* (não teve aumento nos requisitos de cliente)?

   "Fez elicitação através de revisão de requisitos" ou "Falta de protótipos"

6) Por que nos dias 10 e 11 o rendimento de Daniel na sua função de *especificação* teve uma discrepância muito grande (342 *validation* vs 34 *validation*)?

   Influencia negativa

7) Mesmo entregando o projeto com *bugs* não encontrados/corrigidos, qual foi o maior fator contribuinte que permitiu entregar o projeto a tempo?

   Negociação por mais tempo (estender deadline)

8) Identifique os dois fatores mais contribuintes que levaram a falta de Creditos apresentada a partir do dia 11.

   Contratação e treinamento

9) Um funcionário ficou sem nenhuma tarefa durante quatro dias. Quem foi esse funcionário?

   Arden

10) Hora de término: _____

Se tiver alguma sugestão, escreva no verso. Obrigado.