# – Provenance in Games

## 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[[1]](#footnote-1) experience may be useful for understanding the achieved results. A game flow analysis might be fundamental for detecting symptoms of problems that occurred due to wrong decision-making or even bad gameplay 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 player’s understanding of the game flow, 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 flow 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 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 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 flow, and providing the means for a post-game analysis. This chapter explains the *Provenance in Games* framework along with the process of provenance gathering, the provenance graph construction, and the provenance visualization.

This chapter is organized as follow: Section 4.2 provides the mapping of provenance terms to the game context. Section 4.3 provides a storage structure for the information gathered from the game, while Section 4.4 introduces the provenance graph generation and visualization. Lastly, Section 4.5 presents the final considerations of this chapter.

## Data Model

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 can be represented in games. As mentioned at chapter 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, 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 beings capable of making decisions, while *entities* represent inanimate 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 [17] 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 , 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.



Figure 1: Data model diagram. Gray classes represent generic provenance classes.

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, 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 flow 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.

## Provenance Gathering and Data Structure

Our conceptual framework requires a data infrastructure for all the necessary data to be used later for provenance analysis. The storage structure can be similar to lists. For example, each *agent* can have a list of actions that contains all his executed actions. This 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. For events, it is possible to use an analogous approach, storing all events by trigger.

When an action influences another action, executed by another *agent* (for example, a buff spell), then the influenced action will have a pointer 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 action’s details and a pointer 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*. With this structure, it is possible to infer who influenced the action by following the links from 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 a battle between a mage and an orc fighter. At the beginning of the battle, the mage casts a spell called *weakness* over the orc. 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. Due to the distance between the orc and 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. On the next turn, the mage cast another spell (*stonefist*), which only causes damage. In the orc’s turn the orc makes an attack action because he is now in melee range of the mage. However, due to the spell casted by the mage (*weakness*) in the last round, the orc suffers a penalty to his attack roll.

In other words, the attack action from the orc was influenced by the spell from the mage. When this happens, the attack action information that is stored will need to have a link to the spell action. Since a pointer to that particular action was already generated for future usage, that pointer can be consumed (because the effect was only in the next attack) and used to link the actions. If the spell had a duration that affected all attacks until it expires then each turn one pointer will be consumed (but not necessarily bound to another action). When there are no more pointers, the action that generated influences cannot influence another because it expired. If the duration is not actually in turns, but is tied to the times used, the next five attacks for example, then each attack will consume one pointer from the stack. illustrates this example of combat scenario between the mage and the orc, including the influence between the actions of casting spell *weakness* and the orc attack. Without pre-generating the influence pointer, it would be necessary to check all lists in search of the action that influenced it.

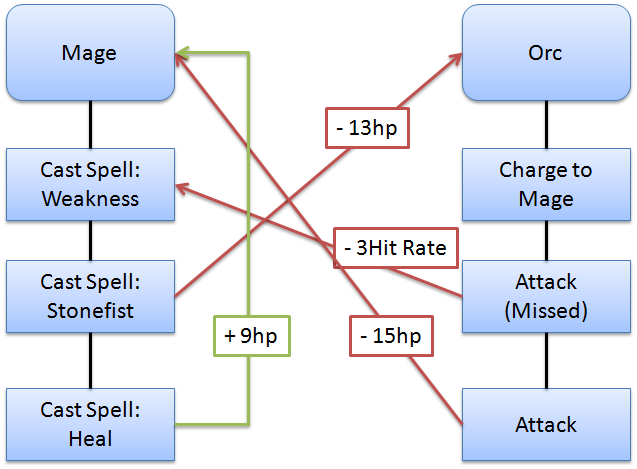


Figure 2: Provenance representation of a combat.

*Agents* present in a scene, or place, can also be represented in a similar way as actions. 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, 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 flow log*[[2]](#footnote-2) that is used for the generation of a provenance graph. However, even using the data structure presented in this chapter, the *game flow 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 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*, which most are there only walking around to give life to the city, it can be collected only from the ones that interacted with or influenced the actions of other *agents* relevant to the story. Doing this way, it filters *agents* that are only there to simulate crowd (PASSOS *et al.*, 2010). Another possible filter is for actions. For example, actions like 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 flow 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 flow 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.

## 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. In the previous sections, the framework to store such information was introduced. However, not all stored information in the *game flow 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 a provenance visualization approach named *Proof Viewer* (Provenance Flow Viewer), which allows the analysis of generated *game flow log* through a provenance graph. A game using the *Provenance in Games* framework is able to generate a *game flow log* that can be displayed by *Proof Viewer*, allowing the user to analyze it. Figure 3 illustrates the relationships between the game, using the framework to generate the *game flow log* from the game session, and Proof Viewer.

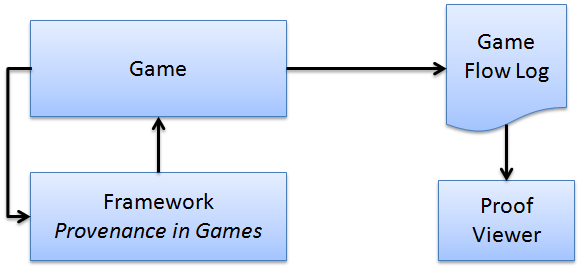


Figure 3: Relationship between a game using the framework and *Proof Viewer*.

At the end of the game session, or at any moment during it, the *game flow log* is generated containing all collected information of the session until that moment. This log is then processed and used to generate a provenance graph. The graph construction is based on the information contained in the log. The graph itself is a visual representation of the *game flow 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 flow log*. The information is extracted from the log and used to created 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 flow 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. In the following sections…

### Shape and Color

Because *Proof Viewer* is a visualization approach for the *game flow log* by means of a provenance graph, it uses certain features to aid the visualization and distinction of the information displayed from the *game flow 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 noted, 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 4. Also in Figure 4, activities are colored as light gray, agents are dark blue, and *entities* are beige and dark gray. Color can also used to distinguish *agents*, *activities*, and *entities* according to their relevance or sub-type. For example, the beige and dark gray *entities* in Figure 4 illustrates that they belong to different types. This is useful because it is possible to distinguishing a player from monsters by using different colors since both types are *agents*, thus have the same shape.



Figure 4: 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 4, 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.

### 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 the 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 (from anything to anything) during the game.

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.

## Final Considerations

This chapter presented a conceptual framework to introduce the usage of provenance in games. The framework, called *Provenance in Games*, provides the necessary mappings of provenance terms into the game elements. It also provides a method for creating a *game flow log* and visually represents it with a provenance graph for a better understanding of the events occurred during a game session. Lastly, it was presented the *Proof Viewer*, an approach for generating a provenance graph from a *game flow log* and aiding the analysis, as well as guidelines on how to structure the information gathered.

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; PEDRYCZ; SWINIARSKI, 1998; FAYYAD; PIATETSKY-SHAPIRO; SMYTH, 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 flow log*. The log is then used in *Proof Viewer* 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.

# References

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

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.

BRISTOL, Edgar H. Pattern recognition: An alternative to parameter identification in adaptive control. *Automatica*, v. 13, n. 2, p. 197–202, mar. 1977. Acesso em: 25 nov. 2012.

CHIALVO, D.R.; BAK, P. Learning from mistakes. *Neuroscience*, v. v. 90, n. 4, p. 1137–1148, jun. 1999. Acesso em: 5 jul. 2012.

CIOS, Krzysztof; PEDRYCZ, Witold; SWINIARSKI, Roman W. *Data mining methods for knowledge discovery*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

CLARK, George. The organization of behavior: A neuropsychological theory. *The Journal of Comparative Neurology*, v. v. 93, n. 3, p. 459–460, 1950. Acesso em: 5 jul. 2012.

DIEHL, Stephan. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. [S.l.]: Springer, 2007.

FAYYAD, Usama; PIATETSKY-SHAPIRO, Gregory; SMYTH, Padhraic. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, v. 17, n. 3, p. 37, 15 mar. 1996. Acesso em: 25 nov. 2012.

FREIRE, J. *et al.* Provenance for Computational Tasks: A Survey. *Computing in Science Engineering*, v. 10, n. 3, p. 11 –21, jun. 2008.

HAN, Jiawei; KAMBER, Micheline. *Data Mining: Concepts and Techniques*. [S.l.]: Morgan Kaufmann, 2006.

NAVARRO, Emily. *Simse: a software engineering simulation environment for software process education*. 2006. California State University at Long Beach, Long Beach, CA, USA, 2006.

PASSOS, Erick Baptista *et al.* A bidimensional data structure and spatial optimization for supermassive crowd simulation on GPU. *Comput. Entertain.*, v. 7, n. 4, p. 60:1–60:15, jan. 2010. Acesso em: 10 jun. 2013.

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

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

WITTEN, Ian H.; FRANK, Eibe. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. [S.l.]: Morgan Kaufmann, 2005.

1. Gameplay is defined as “the total experience provided by a game’s structure and mechanics” (THOMPSON; BERBANK-GREEN; CUSWORTH, 2007). [↑](#footnote-ref-1)
2. The *game flow log* can also be viewed as the player’s journey. [↑](#footnote-ref-2)