# – 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 its complexity, reproducing the same state can be unviable, making it difficult to replay and try new solutions.

This game flow analysis deserve particular attention for serious games (ABT, 1987), which are games used for purposes other than entertainment while still providing pleasure. Serious games have been used for aiding students to learn and understand concepts taught in classrooms (BAKER; NAVARRO; VAN DER HOEK, 2003; NAVARRO, 2006) due to their stimulating curiosity characteristic and for providing motivation for learning (PRENSKY, 2001). Understanding the educational results obtained in a serious game is important to assimilate the knowledge and concepts passed in the game. In addition, examining the game flow allows for the identification of good and bad attitudes made by the player or by developers. This knowledge can be used in future game sessions to avoid making the same mistakes or even to adjust gameplay features.

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 understanding of the story development throughout the game. Watching the game unfold again for a second time by a replay or looking at the 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 his armies that tipped in his favor? Such questions are common to arise and sometimes their influences are not so apparent to the player. Or even if they were, analyzing it in more details might provide useful insights for future occasions.

With this in mind, the goal of this work is to improve the player’s understanding of the game’s flow, providing insights on how the story progressed and the influences in the outcome. In order to improve understanding, this proposed work provides the means of analyzing the game flow by using provenance. The provenance analysis is done by processing the collected gameplay data and generating a provenance graph, which relate the actions and events that occurred during the game session. This provenance graph allows the player, or a third user (ex: tutor), to identify critical actions that influenced the game outcome and helps to understand how events were generated and which decisions influenced them. This process also aids in the identification of mistakes, allowing the player to reflect upon them for future interactions.

The goal of this work is to propose a framework that collects information during a game session and maps it to provenance terms, providing the means for a post-game analysis. Thus using digital provenance (FREIRE *et al.*, 2008b) concepts for representing the game flow. In this chapter it will be explained the *Provenance in Games* framework, along with the process of provenance gathering, the provenance graph construction, and the provenance visualization.

As such, this chapter is organized as follow: BLABLABLA

## 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 to elements that can be represented in games. As was mentioned in the previous chapter, the *Open Provenance Model* and PROV use three types of vertex: *Artifacts/Entities*, *Process/Activities* and *Agents*. In order to map these vertex types, it is first necessary to find their counterparts in the game context. To avoid misunderstanding, throughout this chapter it will be adopted the terms used in PROV for vertices (*entities*, *activities*, and *agents*).

Starting with *entities*, their provenance definition states that they are physical or digital objects. This definition already gives a clue which role *entities* can represent in the game context: objects. An object can be anything used in the game, for example in the case of an RPG, *entities* can represent weapons, potions, legendary artifacts, magical objects, etc. It can represent anything meaningful to the development of the game history or even objects in a scene that someone interacted with. On the other hand, *agent* definition is a person, an organization, or anything with responsibilities. In the game context, *agents* can be mapped as people present in the game: 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. Lastly, *activities* according to its definition are actions taken by *agents* or interactions with other agents or *entities*. So, in a game context, *activities* can be viewed as actions or events executed throughout the game, like attacking, dodging, and jumping.

Now, 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, but also provides rules to extend these relationships to be more suitable to a game context. Also, the PROV model can deal well with the aspect of time, which can be heavily explored in games, especially on games focused on storytelling, recording when each event happened and using this information to generate other events without compromising continuity.

To generate actions and control events, each NPC in the game requires a behavior controller in order to control his actions, providing an array of behavior possibilities. It can be of any time, as long as the information is recorded when executed. Event triggers are also analogous. The information extracted is later used in the provenance graph, so it is recommended to store relevant data. Relevancy varies from game to games but ideally it is any information that can be used to aid during analysis.

Actions can be represented by a series of attributes that describe it and the context it was involved, 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, what the action produced, and what its affect should be recorded for analysis.

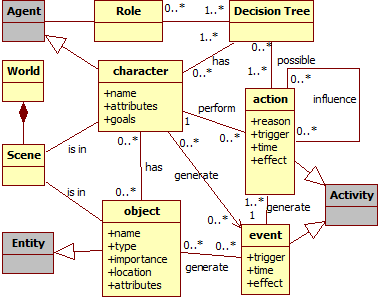


Figure 1: Data model diagram. Gray classes represent provenance classes.

For example, a monster attacked the player and scored a hit causing some damage, which in turns decreases the player’s hit points (hp). For this action, the relevant information is: when it was executed (time, turn, or combat round), who executed it (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 is the player), and the consequences of this action (decreased the player’s hp). If the action affects more than one person, then record all people involved and how the action affected each one. For example, the attack action was actually a buffing attack, which provides a boost the monster’s allies and does damage to the target. So aside from recording the damage inflicted, 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 persons. 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. illustrates this data model diagram, which maps provenance types to the context used in games. The gray classes are the original provenance objects used for the mapping. The diagram also exemplifies some basic information that should be considered to be stored.

## Provenance Gathering and Data Structure

In order to store all the necessary data to be used later for provenance reasons, it is required a storage structure. Depending on the information structure, it is possible to use the structure itself for inference in provenance, simplifying some unnecessary information. 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 say who was responsible for the execution of the action. For events, it is possible to use an analogous approach, storing all events by trigger.

In the case of the action having an external influence from another action executed by another *agent* (for example, a buff spell), then the action will have a pointer to the action that influenced it. If an ally used a buff spell on the player that buff his attack rate then, when the player’s attack action is generated, it will be stored the action’s details and a pointer to the ally’s action that provided the buff. There will be no need to explicitly mention the ally, only his action 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 influenced action. If there were multiple influences in the executed action, then it is necessary to have a pointer for each action that influenced it. In the case of an action that generates influence, it is necessary to temporarily save a pointer to this action for future actions that might be affected by it, while also indicating which types of action can be influenced.

For example, imagine a battle between a mage and an orc fighter. At the beginning of the battle, the mage cast a spell called *weakness* in 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 charges 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 turn, 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, then that pointer can be consumed (because the effect was only in the next attack) and used to link the actions. Without pre-generating the influence pointer, it would be necessary to check all lists in search of the action that influenced it. If the spell had a duration that affected all attacks until it expires, then it can be used a stack of pointers, where each turn one pointer will be consumed (but not necessarily bound to another action). When the stack empties, then that action cannot influence another because it expired. If the duration is not actually in turns, but it is tied to the times used, the next five attacks for example, then each attack will consume one pointer from the stack and used to bind the actions. illustrates this combat scenario between the mage and the orc, including the influence between actions.

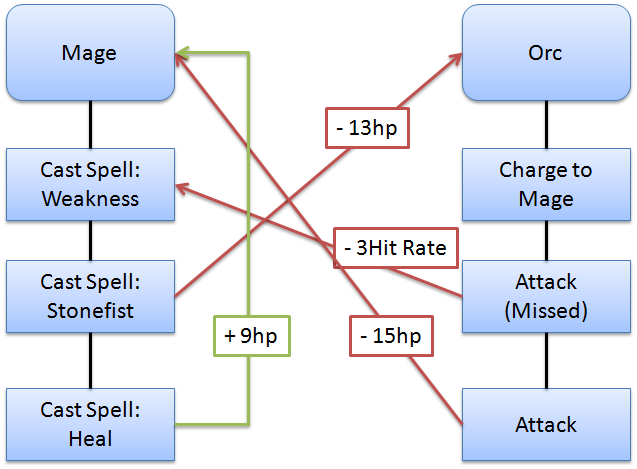


Figure 2: Provenance representation of a combat.

*Agents* present in a scene, or place, can also be represented in a similar way as actions. Each scene will have a list of *entities* and *agents* that were in it, which in turn contain a list of actions executed while in the scene. Since the sequence of actions in a game is important, situations where the same place was visited multiple times throughout the game, like a city, might cause complications in graph, making it difficult to understand. To avoid this, instead of putting everything in the same scene, treat each visit to the place 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, break it apart and treat each visit as a different place, or instance. This is reasonable because even if it is the same city, it was visited at different times. This way will result in a clearer visualization of the player’s journey and interactions.

All these collected information made throughout the game, as they were executed, composes the *game flow log*[[2]](#footnote-2) that is used for the generation of a provenance graph. However, even using this storage structure, the *game flow log* can 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 will be stored might be useful for reducing the provenance graph size. Depending on how the filtering is done, no relevant information from the game will be lost.

For example, depending on the place, the number of *agents* present in it can be astronomical. Such is the case in cities or sometimes villages. 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*. Doing this way, it will filter the *agents* that are only there to simulate a crowd. 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. Filtering these types of non-essential actions or *agents* will decrease the quantity of information gathered, 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. While the game session is running, minor filters can be are used to reduce the *game flow log* size and when the session is over, apply other types of filters to reduce even more the size of the log. The more irrelevant information removed in this stage, fewer inferences will be required during the graph visualization in order to clear the graph from unnecessary information. This way the user will be able to devote more of his attention to analyzing 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, it was introduced the framework to store such information. However, not all stored information in the *game flow log* is relevant for the analysis desired, even pre-filtering the information before processing the graph. These elements can act as noise and can be omitted during provenance analysis by inferences, since some actions might not be relevant for one type of analysis but essential for another type.

This section introduces a provenance visualization tool named *Proof Viewer* (Provenance Flow Viewer), which allows for 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 analyzed by *Proof Viewer*. Figure 3 illustrates the relationships between the game, using the framework to generate the *game flow log* from the game session, and Proof Viewer.

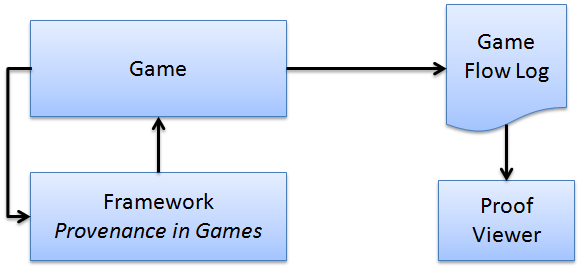


Figure 3: Relationship between a game using the framework and *Proof Viewer*.

At the end of the game session, or at any moment during it, the *game flow log* is generated containing all collected information throughout the session. This log is then processed and used to generate a provenance graph. The graph construction is based on the information contained in the log and it is a visual representation of the *game flow log*, allowing the user to interact and analyze the information collected from the game session, aiding him to reach decisions about how the events in the game occurred and how they affected the outcome. The graph also allows the visualization of the consequences that each action generated, 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 their respective visual representations in the graph by either being vertices or 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 identify them. These inferences can be done by collapsing chains of actions that are between them to find indirect influences. Omitting facts can also be used to remove unnecessary or irrelevant information that came with the *game flow log* for a better understanding and clearer visualization of what is relevant for the analysis. No information is lost in this process, so it is possible to undo all changes made during analysis.

### Shape and Color

Because *Proof Viewer* is a visualization of the *game flow log* by the means of a provenance graph, it uses certain features to aid in 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 of the information. For example, vertices can use their timestamps and names as labels while edges use their type of influence (ex: damage, healing, buff).

As previously noted, vertices can have different shapes according to their types. *Activities* are represented by squares, *entities* by circles, and *agents* by 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 did not interact with other *activities* can be 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, like the beige and dark gray *entities* in Figure 4. This is useful because it is possible to distinguishing a player from monsters by using different colors since both types are *agents*, thus having the same shapes.

Different formats can also be used for edges, as well as colors, to distinguish them. 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. It is also possible to make the edge to be dashed in order to emphasize their importance, or lack of. Also in Figure 4, neutral edges are dashed to emphasize their lack of importance.

Despite mentioning only *activities*, *entities*, and *agents*, it is possible to create other types of vertex for the graph in order to better organize it. For example, creating a vertex type to represent locations from the game and bind all actions that took place in each location, as well as instances of the agents that were there. Also, representing the player’s journey in the game by linking each location according to order it was visited.

### Filters



Figure 4: Example of a generated provenance graph.

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 during analysis in the provenance graph through filters. These filters can be of three types: vertex filter, edge filter, and status filter. The first two filters are related to the graph, omitting vertices and edges. The last one, status filter, alters the way the information is displayed. For example, to analyze the attribute hp, both from monsters and players, using the status filter will change 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 together and thus changing the graph visualization granularity. For example, instead of displaying information in a daily basis, it is possible to group them together in order to display the summary of the events in a weekly scale. Another usage of collapse is to group *activities* from the same *agent*, making easier to see all influences and changes that the *agent* did throughout the game. Figure 5 illustrates a collapse by grouping an *agent’s activities* with the *agent*. Note the vertex size difference when collapsed. 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 filter is the status filter. 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, yellow, or green color. As an example, imagine that we desire to analyze the player’s hit points (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.

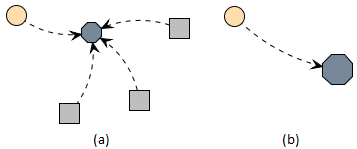


Figure 5: Collapsing Vertices

Graph containing an *entity*, an *agent*, and three *activities* (a), and the collapsing of the agents’ *activities* into a single vertex (b).

Depending on the game played, a game session might take several hours to complete, or even days in case of RPGs. This will make the size of the provenance graph to be overwhelming to the user, even with pre-filtering the *game flow log* before its use. One way to avoid such situations is to show the provenance graph already filtered instead of its full extension. For example, before showing the graph to the user, make some collapses to reduce its size. Combats can be identified and collapsed into single vertex for each instance. Places visited in the game can also be collapsed into single vertex, containing all interactions made in that location, even combats. There is no problem of having collapses inside collapses, in this case, a collapsed combat inside a collapsed area visited by the player, which contains other actions aside from the combat like interacting with the ambient. This will give an impression of a map from the player’s journey throughout game, showing vertices for each location visited by the player, while allowing the player to expand for analysis only the places he desires. Similar to *google maps*, where it shows the entire world and allows the user to zoom in specific locations. However in this case, it will show instances of the journey taken by the player.

It is also possible to go beyond that. Instead of collapsing all combats and locations, collapse only those that 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, so the battle outcome is already known and can be used to decide if it was 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.

## Final Considerations

This chapter presented a framework to bring provenance to games. The framework, called *Provenance in Games*, provides the necessary mapping of provenance term to game elements. It also provides a method of creating a *game flow log* and analyzing it with a provenance graph for a better understanding of the events occurred during a game session. Lastly, it was presented the *Proof Viewer*, a tool for generating a provenance graph from a *game flow log* and aiding the analysis.

Currently, Proof Viewer does not provide inference for the user, only the means necessary to infer. The game developers will need to create inference rules customized to their games to aid in the analysis, such as clustering sequences of actions and identifying 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 sensitive. 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* 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.

# 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. Accessed: 25 nov. 2012.

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

KOHWALTER, Troy; CLUA, Esteban; MURTA, Leonardo. Provenance in Games. In: 2012 XI BRAZILIAN SYMPOSIUM ON GAMES AND DIGITAL ENTERTAINMENT (SBGAMES), nov. 2012, Brasilia. *Anais*... Brasilia: In: XI SBGames, nov. 2012.

LINDLEY, Craig A. Narrative, Game Play, and Alternative Time Structures for Virtual Environments. In: GÖBEL, STEFAN *et al.* (Org.). *Technologies for Interactive Digital Storytelling and Entertainment*. Lecture Notes in Computer Science. [S.l.]: Springer Berlin Heidelberg, 2004. p. 183–194. Available: <http://link.springer.com/chapter/10.1007/978-3-540-27797-2\_25>. Accessed: 9 abr. 2013.

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

OXLAND, Kevin. *GAMEplay and Design*. [S.l.]: Addison-Wesley, 2004.

PREMIS WORKING GROUP. *Data Dictionary for Preservation Metadata*. Technical, no Final Report of the PREMIS Working Group. Preservation Metadata: Implementation Strategies (PREMIS), 2005. Available: <http://www.oclc.org/research/projects/ pmwg/premis-final.pdf>.

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

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

1. In this work, gameplay is defined as the player’s experiences throughout the game session or the game’s rules and challenges. [↑](#footnote-ref-1)
2. The *game flow log* can also be viewed as the player’s journey. [↑](#footnote-ref-2)