Game Flux Analysis with Provenance

Troy C. Kohwalter, Esteban G. W. Clua, and Leonardo G. P. Murta

Instituto de Computação, Universidade Federal Fluminense, Niterói – RJ, Brazil
{tkohwalter,esteban,leomurta}@ic.uff.br

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 fluxes of a concrete game play may be useful for different reasons: understanding problems related to gameplay, data mining of specific situations, and even understanding educational and learning aspects in serious games. We introduce a novel approach based on provenance concepts in order to model and represent a game flux. We model the game data and map it to provenance to generate a provenance graph for analysis. As an example, we also instantiated our proposed conceptual framework and graph generation in a serious game, allowing developers and designers to identify possible mistakes and failures in gameplay design by analyzing the generated provenance graph from collected gameplay data.

Keywords: Game flux, Game analysis, Provenance, Graph Analysis.

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 fluxes of a concrete gameplay experience may be useful for understanding the achieved results. A game flux analysis may also be fundamental for detecting symptoms of problems that occurred due to wrong decision-making or even bad gameplay design. Besides that, without any formalized process, this type of analysis may be subjective and, depending on the game dynamics and its complexity, it would require playing the game successively, making the same decisions, to intuitively guess which ones were responsible for generating the observed effects. Thus, reproducing the same state can be unviable, making it difficult to replay and identify, in a trial and error approach, the source of the problem. In addition, examining the game flux might allow for the identification of good and bad attitudes made players. This knowledge can be used in future game sessions to avoid making the same mistakes or even to adjust gameplay features.

The analysis process for detecting gameplay issues is done nowadays in an artisanal way by using a popular beta testing [1] approach. The beta test phase is an indispensable source of data for the developers about technical issues or bugs found in the game. Normally, beta testers are volunteers who were recruited to play the game in an early, pre-release, build of the game where they can provide information about

D. Reidsma, H. Katayose, and A. Nijholt (Eds.): ACE 2013, LNCS 8253, pp. 320–331, 2013.

technical issues and provide feedback about the gameplay mechanics. Thus, beta testing is a crucial part of the development to identify important issues in the game. However, developers have little control over the beta testers' gameplay experience or the environment because they can play at home.

The goal of this paper is to improve the game designer's understanding of the game flux, providing insights on how the gameplay progressed and influences in the outcome. In order to improve understanding, we provide the means to analyze the game flux by using provenance. The provenance analysis requires processing the collected gameplay data and generating a provenance graph, which relates the actions and events that occurred during the game session. This provenance graph allows the user to identify critical actions that influenced the game outcome and helps to understand how events were generated and which decisions influenced them. This analysis could be used in conjunction with the beta testing in order to aid in the identification of gameplay or technical issues, allowing the designer to analyze the tester's feedback report and the gameplay data from the game session.

In our previous work [2], we introduced the usage of digital provenance [3] in games. The main goal of the previous work was 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. This was the first time that the provenance concept and formalization was used in the representation of game flux. The present paper is based on the conceptual framework definition introduced in the previous paper. However, while in the previous work focused on the provenance gathering, this work focuses on the provenance graph construction and manipulation to support analysis. Even though the scenario used in this paper 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 and balancing, data mining, and even for storytelling.

This paper is organized as follows: Section 2 provides related work in the area of game flux analysis. Section 3 provides a background on provenance, and Section 4 introduces our framework for provenance gathering. Section 5 presents our approach for provenance visualization through graphs. Section 1 presents the adoption of provenance visualization in a software engineering game, with visualization examples. Finally, Section 7 concludes this work and points out some future work.

2 Related Work

In the digital game domain, Warren [4] proposes an informal method to analyze the game flux using a flux graph, mapping game actions and resources into vertices. By his definition, resources are dimensions of the game state that are quantifiable, while actions are rules of the game that allowed the conversion of one resource into another. Consalvo [5] presents a more formal approach based on metrics collected during the game session, creating a gameplay log to identify events caused by player choices. *Playtracer* [6] allows the user to visually analyze play steps, providing detailed visual representation of the actions taken by the player through the game.

These three approaches are developer-oriented, meaning that they aim to improve the quality of the game by providing feedback to the development team. However, Consalvo [5] presents a template for analysis, acting as guidelines to how the analysis should be done. Meanwhile, *Playtracer* [6] focuses on identifying the player's strategies by visually analyzing play traces instead of using queries.

Lastly, the *Game Analytics* [7] from Unity3D [8] allows visualizing game data as heat maps directly on the scene. This identifies bottlenecks and hotspots, and underused and overused areas of the game. It also measures the game retention rate, where players are stopping playing, and how the game progression develops.

3 Provenance

Provenance is well understood in the context of art or digital libraries, where it respectively refers to the documented history of an art object, or the documentation of processes in a digital object's life cycle [9]. In 2006, at the *International Provenance and Annotation Workshop* (IPAW) [10], the participants were interested in the issues of data provenance, documentation, derivation, and annotation. As a result, the *Open Provenance Model* (OPM) [11] was created during the *Provenance Challenge* [12], which is a collocated event of IPAW. Recently, another provenance model was developed, named PROV [13], which can be viewed as the successor of OPM. Both models aim at bringing provenance concepts to digital data.

Both provenance models assume that provenance of objects is represented by an annotated causality graph, which is a directed acyclic graph enriched with annotations. These annotations capture further information pertaining to execution. According to [11], a provenance graph is the record of a past or current execution, and not a description of something that could happen in the future. The provenance graph captures causal dependencies between elements and can be summarized by means of transitive rules. Because of this, sets of completion rules and inferences can be used in the graph in order to summarize the information.

4 Provenance Gathering in Games

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 in section 3, 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 paper 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 [14] 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 Fig. 1, 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.

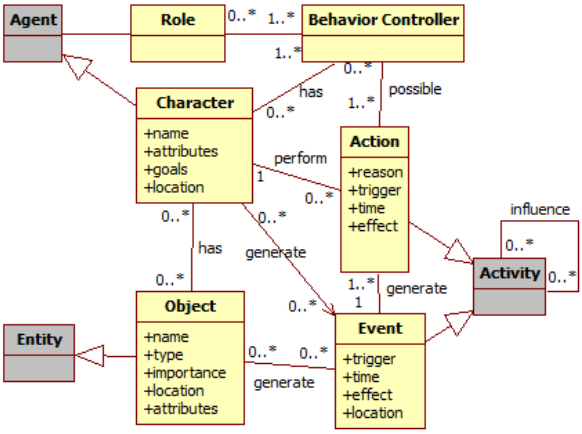

Fig. 1. Data model diagram. Gray classes represent generic provenance classes.

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.

5 Provenance Visualization

The purpose of collecting information during a game session is to be able to generate a provenance graph to aid the developer to analyze and infer the reasons of the outcomes. In this paper we introduce a provenance visualization tool named *Prov Viewer* (Provenance Flux Viewer), which uses JUNG [15] graph framework and allows detailed analysis of a previously gathered game flux log through a graph. A game using the *provenance in games* conceptual framework is able to generate *a game flux log* that can be analyzed by *Prov Viewer*.

First, the *game flux log*, which contains game events, is processed and used to generate a provenance graph for analysis. After that, our tool creates the graph's edges and vertices following a defined set of rules to build the provenance graph. This graph is a representation of the *game flux log* and is available for the developer to interact and analyze, reaching events and causes about how events occurred during the game, and how they influenced other events. It is also possible to manipulate the graph by omitting facts and collapsing chains of action for a better understanding and visualization experience. No information is lost in this process, so the user can undo any changes made during analysis.

Fig. 2. illustrates a small example of a generated provenance graph. Following the provenance notation specification, each vertex shape of the example is related to its type. Square vertex represents *activities*, circles represent *entities* and octagons represent *agents*. The edges in the provenance graph represent relationships between vertices, which can be *agents*, *entities* or *activities*. As such, *activities* vertices can be positively or negatively influenced by other *activities* and have relationships with *entities* and *agents*. The context of such relationships may vary according to the type of relation between vertices.

Prov Viewer has other features besides vertex shape by type. It uses shapes and colors to distinguish displayed information and provides two types of filters: vertex and edge filter. As previously noted, vertices have different shapes according to their types. However, it is also possible to differentiate one vertex from another with different borders and colors. As an example, *activities* that do not interact with other *activities* are dotted, as illustrated in the upper right corner of Fig. 2.

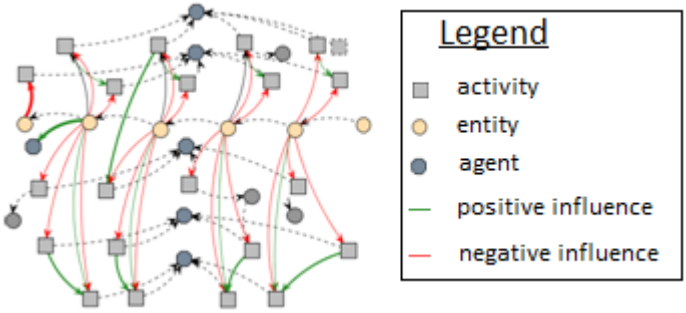

Fig. 2. Example of a generated provenance graph

Different formats can also be used for edges, as well as colors. The thickness shows as how strong the relationship is. A thin edge represents a low influence on the *activity*. On the other hand, the higher the influence is, the ticker the edge. 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 Fig. 2, where neutral edges are dashed to emphasize their lack of importance.

In order to better analyze graph data, the vertex filter feature is also available. 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. These elements act as noise and can be omitted during analysis. To do so, it is possible 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 granularity. Another usage of collapse is to group *activities* from the same *agent*, improving visibility of all influences and changes that the *agent* did throughout the game. Similar edges that have the same target are also grouped together when collapsing vertices, as shown in Fig. 3. The collapsed edge's information is calculated by the sum or average (depending on the edge type) of the values from the collapsed edges. For example, edges representing expenses uses sum, while edges representing aid modifiers (in percentage) uses average. Another type of filter present in *Prov Viewer* is the edge filter, which filters edges by context and by the type of relationship.

The last feature present is the attribute status display. When selecting the desired attribute, all vertices with the specified status have their colors changed according to their respective values. It uses the traffic light scale [16], which indicates the status of the variable using three colors: red, yellow, and green. As an example, imagine that

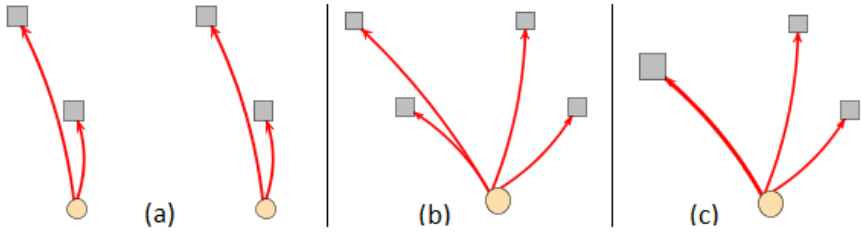

Fig. 3. Collapsing vertices. The original state showing four *activities* and two *entities* with edges of the same type (a), the collapse of both *entities* into one (b), and the collapse of two *activities*, and their respective edges since they were from the same type (c). The size of the resulting edge is bigger than the original ones as a resulting from summing each edge's values.

we desire to analyze the player's financial situation throughout the game. When filtered by this status, all vertices that contain a player financial value have their colors changed according to their values. Activating this type of feature allows the developer to see the player's finances throughout the game, making it easier to identify situations where he might have had financial problems (red color). Section 1 provides more examples of those features.

Using these features for graph manipulation and visualization, the developer is able to interact with the provenance graph, identifying relevant actions that had an impact in the story or in the desired type of analysis. It can also be used to analyze player's behavior, detecting situations that the player had difficulties or didn't behave according to the developer's plan. The next subsection describes alternatives to deal with problems related to graph size and visualization.

5.1 Granularity

Depending on the game style, a game session might take several hours or days to be completed. This makes the size of the provenance graph overwhelming, even when making pre-filtering during the generation of the *game flux log*. One way to avoid such situations is to show the provenance graph with some filters selected instead of its full extension. For example, before showing the graph to the user, it is possible to collapses less relevant vertices to reduce the graph's size. For instance, combats stages 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. It is also possible to collapse collapsed vertices. 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 developer to expand only the situations he desires to analyze.

Instead of collapsing all combats and locations, filters can be used to 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 developer. This is possible because provenance is analyzed from the present to the past. This way, com-

bats 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 may be interesting to display all actions for a correct analysis, allowing the player to identify important facts that influenced the combat outcome. Note that this type of filter is heavily dependable of the game context, so a specific set of filters should be implemented for each individual game.

6 Case Study

We instantiated this provenance analysis infrastructure, which uses the proposed framework presented in [2] and described in section 4, in a Software Engineering educational strategy game named SDM (Software Development Manager) [17]. The goal of SDM is to allow undergraduate students to understand the existing cause-effect relationships in the software development process. Thus, the adoption of provenance becomes an important instrument to better support knowledge acquisition, allowing tracking mistakes made during a game session or identifying game mechanics that requires tinkering.

6.1 Information Storage

The information structure used on SDM is similar to the one explained in [2]. As such, each project contains a list of employees that are involved in its development. Each employee has a list of actions executed as well as links to other actions in case of external influences. Throughout the game, information is collected and stored for generating the provenance graph used during analysis. Since provenance graphs contains three types of vertex (*activities*, *agents*, and *entities*), the collected information is mapped to each type, according to the data model explained in [2]. Each vertex contains different information according to its type.

Activities vertices, which represent actions executed during the game by employees, store information about their executions. This information includes who executed it, which task and role the employee was occupying, as well as the current morale and stamina status. Worked hours, credits spent to execute the action, and progresses made are also stored. Besides those, if the action had any external influences or used or altered entities, then links to them are also stored.

Agent vertices, representing employees, store the employee's name, his current staff grade, his level, human attributes, which are used in the game, and specializations. *Entities* vertices represent *Prototypes*, *Test Cases*, and instances of the *Project's* development. All information from the game is collected in real time, during the execution of actions and events. Thus it is required to change the method responsible for executing each action and event to also store information. After the data is collected and extracted, a provenance graph corresponding to that scenario is generated and displayed for visual analysis, similarly to the one presented by Fig. 2.

6.2 Provenance Graph

With the adaptations made in the original SDM [2], it is possible to collect data and use it to generate a provenance graph. The collected game data, known as *game flux log*, is exported to *Prov Viewer*. In that application, the data is processed and used to generate a provenance graph to aid in the analysis process.

By analyzing the graph, it is possible to reach some conclusions of why the story progressed the way it did. As an example[1], Fig. 4 illustrates a scenario where the player had financial problems. To simplify the picture, some collapses were made, omitting most of the *agent's activities*. The *entities* represent instances of the development stage and are colored according to the player's financial condition. The *activities* present in the picture represent hiring actions in gray and resigning actions in brown.

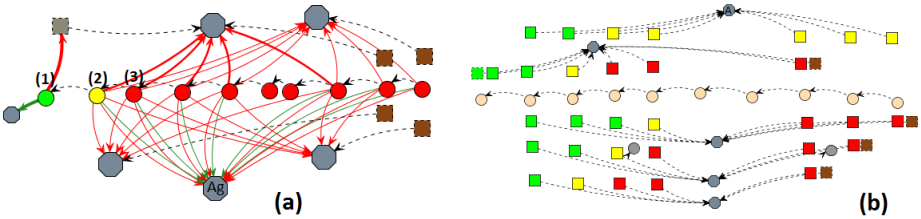

Fig. 4. An example of credits status filter (a) and the non-collapsed provenance graph (b) using filter: Morale. Brown activity vertices represent employees leaving the player's staff.

Fig. 4 was already subject of an attribute status display and a filter to show the player's credits status, both in the edges and in the vertices. In vertex 1, the project had a substantial financial income and a new employee was hired, as marked by the thick green edge to an *agent* and thick red edge to a gray dotted *activities*. The player's credits are also in a green zone as marked by the project's vertex color. However, due to the hiring fee paid in vertex 1 and the resources used by the staff in vertex 2, the player's credits changed to a yellow zone, even with the minor income from *agent* Ag.

In vertex 3, the player's credits changed to red zone due to payments, meaning that his resources are almost empty and he will not have enough credits to keep paying his employees. When that happens, employees' morale is lowered due to the lack of payment and if it reaches red zone, they can resign, as shown by brown *activities*. Observing Fig. 4, we can see the employees' morale getting lower by lack of payment. This helps us to understand why they resigned. Without credits to hire new employees and without a staff, the player loses the game.

This analysis can be used to detect player's behaviors and the reasons of why they lost the game. In the example, the cause was the lack of resources due to hiring a new employee. If it was necessary to hire a new employee, then there is a problem that requires immediate attention. However, hiring an employee causes the player to lose

[1] In order to reduce graph size and provide a quicker understanding for the presented examples, some in game parameters were configured to allow faster state transitions.

the game, leading to the conclusion that if hiring is optional, then some changes might also be required because the penalty is too severe and causes the player to lose, instead of giving only a small setback.

Another example of analysis is checking the employee productivity and understanding why variations occurred by using multiple filters to test theories. In SDM, productivity is defined by the executed task, the amount of outside help, the employee's job (junior, mid-level, and senior), the working hours, and the stamina and morale stats. Fig. 5 illustrates an example scenario. To simplify the graph visualization due to size limits, we focus only on two *agents* and the *entity* known as "project". Those *agents'* roles are programmer and manager, with the manager acting as a supporting role for the programmer.

Analyzing the picture we can see that the programmer's productivity fluctuated throughout vertices 1 to 7. We can also see that the manager did not cause this fluctuation, since his aid bonus did not have much variation. In vertex 2, the programmer maximizes his productivity at the cost of quality. This information, as well as other details about the vertex, is displayed in the vertex's tooltip. The change in vertex 3 can be identified by observing his working hours, which can be done by looking at each individual vertex or by adding a filter, as shown in Fig. 5.

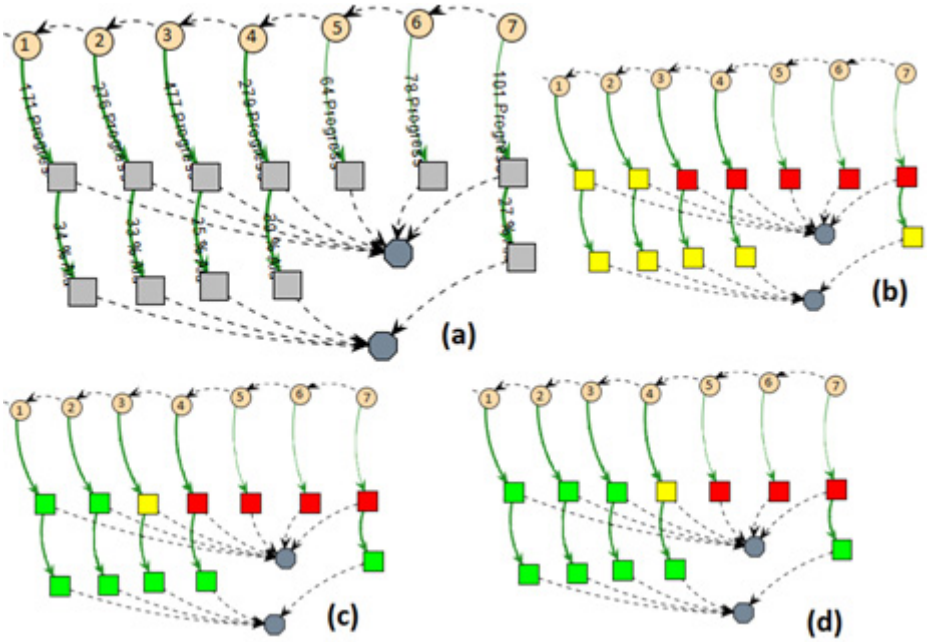

Fig. 5. Example of a provenance graph analysis. The entity is project's stages of the development. Agents are employees from the development staff, with the programmer being the upper agent and the manager the lower one. Graph using different display features: default mode (gray activity vertices) displaying edge values (a), working hours (b), stamina (c), and morale visualization modes (d). Red vertices represent low, yellow represent average, and green represent high values.

In Fig. 5 one can see via the change from yellow to red that the programmer's working hours per day increased. Since the *activity* in vertex 3 is red, it means the employee is doing extra hours, which increases his productivity. From vertices 3 to 7, his working hours remained unaltered. Therefore, the change from vertices 2 to 3 was mainly due the change on his daily working time. However, if we look at vertex 4, we can see a drop in his productivity.

By changing the filter again to show stamina levels, we can see that in vertex 3 the programmer's stamina dropped to yellow because of the extra hours and in vertex 4 it reached red due to exhaustion. Another side effect of his exhaustion was the change on the programmer's morale, which also reached the red zone in vertex 5. Lastly, the small variation from vertices 5 to 7 comes from a random range modifier during productivity computation, since the programmer is already working at minimal levels at the current configuration. With both the morale and stamina at lowest levels, the extra hours were not compensating his productivity loss. As previously shown, if his morale levels do not increase, the programmer might resign. This example of analysis covered all possibilities that affect a programmer's behavior and can be used to further refine game modifiers or state transitions, as well as identifying odd behaviors caused by game modifiers.

7 Conclusion

This paper introduces new perspectives on gameplay modeling and analysis, leveraging the current state of the art, based on gameplay, to a level where the game provenance can aid the detection of gameplay issues. This knowledge can help on (1) confirming the hypotheses formulated by the beta tester, (2) supporting developers for a better gameplay design, (3) identifying issues not reported by testers, and (4) data-mining behavior patterns from individual sessions or groups of sessions.

The provenance visualization can happen on either on-the-fly or post-mortem sessions. It allows the discovery of issues that contributed to specific game fluxes and results achieved throughout the gaming session. This analysis can be used on games to improve understanding of the game flux and identifying actions that influenced the outcome, aiding developers to understand why events 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 in the following game genres: role-playing, strategy, and simulation.

Currently, we do not make inferences to the user, but let the user or developers decide what needs to be inferred. 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. Another interesting research is to automatically identify patterns in the game flux. Lastly, we are working on different graph visualization layouts and also studying the possibility of using game provenance in educational digital games to aid in the understanding of the concepts taught in the game.

Acknowledgments. We would like to thank CNPq, FAPERJ, and CAPES for the financial support.

References

1. Davis, J., Steury, K., Pagulayan, R.: A survey method for assessing perceptions of a game: The consumer playtest in game design. Game Studies 5 (2005)
2. Kohwalter, T., Clua, E., Murta, L.: Provenance in Games. In: Brazilian Symposium on Games and Digital Entertainment (SBGAMES) (2012)
3. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for Computational Tasks: A Survey. Computing in Science Engineering 10, 11–21 (2008)
4. Warren, C.: Game Analysis Using Resource-Infrastructure-Action Flow, http://ficial.wordpress.com/2011/10/23/game-analysis-using-resource-infrastructure-action-flow/
5. Consalvo, M., Dutton, N.: Game analysis: Developing a methodological toolkit for the qualitative study of games. Game Studies 6 (2006)
6. Andersen, E., Liu, Y.-E., Apter, E., Boucher-Genesse, F., Popović, Z.: Gameplay analysis through state projection. In: Foundations of Digital Games (FDG), pp. 1–8 (2010)
7. Wulff, M., Hansen, M., Thurau, C.: GameAnalytics For Game Developers Know the facts Improve and Monetize, http://www.gameanalytics.com/
8. Higgins, T.: Unity - 3D Game Engine, http://unity3d.com/
9. PREMIS Working Group: Data Dictionary for Preservation Metadata. Implementation Strategies (PREMIS), OCLC Online Computer Library Center & Research Libraries Group (2005)
10. Moreau, L., Foster, I., Freire, J., Frew, J., Groth, P., McGuiness, D.: IPAW, http://www.ipaw.info/
11. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., den Bussche, J.V.: The Open Provenance Model core specification (v1.1). Future Generation Computer Systems 27, 743–756 (2007)
12. Miles, S., Heasley, J., Szalay, A., Moreau, L., Groth, P.: Provenance Challenge WIKI, http://twiki.ipaw.info/bin/view/Challenge/
13. Moreau, L., Missier, P.: PROV-DM: The PROV Data Model, http://www.w3.org/TR/prov-dm/
14. Moret, B.: Decision Trees and Diagrams. ACM Computing Surveys (CSUR) 14, 593–623 (1982)
15. O'Madadhain, J., Fisher, D., Nelson, T.: JUNG: Java Universal Network/Graph Framework, http://jung.sourceforge.net/
16. Diehl, S.: Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer (2007)
17. Kohwalter, T., Clua, E., Murta, L.: SDM – An Educational Game for Software Engineering. In: Brazilian Symposium on Games and Digital Entertainment (SBGAMES), pp. 222–231 (2011)