”Bloody Hell! Why did this happen?” Modeling Game Experiences with Provenance

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

**Keywords.** Game flow; Game analysis; Provenance; Learning.

1. Introduction

The conclusion of a game session derives from a series of decisions and actions made throughout the game. In many situations, analyzing and understanding the events, mistakes, and flows of a concrete gameplay experience may be useful for understanding the achieved results. A game flow analysis may also 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.

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 the identification of good and bad attitudes made by the player. This knowledge can be used in future game sessions to avoid making the same mistakes or even to adjust gameplay features.

The goal of this paper is to improve the player’s understanding of the game flow, providing insights on how the story progressed and influences the outcome. In order to improve understanding, we provide the means to analyze 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 user 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.

In our previous work [1], we introduced the usage of digital provenance [2] 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 flow. The present paper is based on the framework definition introduced in the previous paper. However, while in the previous work we introduced the provenance gathering, this work introduces the provenance graph construction and analysis methodology. Even though the main application of provenance used in this paper is over a serious game, we believe that the concepts discussed in this paper are applicable to other kinds of games and useful to support advanced analysis, such as gameplay balancing, events and behaviors data mining, and even storytelling enhancements.

This paper is organized as follows: Section 2 provides related work in the area of game flow 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 6 presents the adoption of provenance visualization in a software engineering game, providing visualization examples. Finally, Section presents the conclusions of this work and points out some future work.

1. Related Work

In the digital game domain, Warren [3] proposes an informal method to analyze the game flow using a flow graph, mapping game actions and resources to vertices. By his definition, resources are dimensions of the game state which are quantifiable, while actions are rules of the game that allowed the conversion of one resource to another. Consalvo [4] presents a more formal approach based on metrics collected during the game session, creating a gameplay log to identify events caused by player choices. Another method, called *Playtracer* [5], offers a way to visually analyze play steps, providing detailed visual representation of the actions taken by the player through the game.

Besides [3], which is superficially described in a blog, the other two methods are developer-oriented, meaning that they aim to improve the quality of the game by providing feedback to the development team. Due to that, we could not find any concrete solution to provide feedback to the player.

Another method that analyzes a story in the field of interactive storytelling [6] was presented in [7]. This method organize the story using PNF networks [8], representing the temporal structure of the events that make up the plot. This structure can also be used in the generation of new events to the story, but is restricted to temporal coherence between the game events, without providing insights of positive or negative reinforcements.

1. 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. In 2006, at the *International Provenance and Annotation Workshop* (IPAW) [9], the participants were interested in the issues of data provenance, documentation, derivation, and annotation. As a result, the *Open Provenance Model* (OPM) [10] was created during the *Provenance Challenge* [11], which is a collocated event of IPAW. Recently, another provenance model was developed, named PROV [12], which can be viewed as a continuation of the 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 [10], 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 is composed of vertices that can represent *artifacts*, *processes*, and *agents* in OPM or *entities*, *activities*, and *agents* in PROV. *Entities* in PROV (and similar to *artifacts* in OPM) represent physical or digital objects like a document, the web, or material objects. *Activities*, which are similar to *processes* in OPM, are actions taken to change or interact with *entities* or *agents*. Lastly, an *agent* (in both models) is a person, software, organization, or *entities* that have responsibilities. The edges of the graph represent a causal dependency between the source, which is the effect, and the destination, which is the cause.

Finally, OPM and PROV have defined the notion of a provenance graph based on a set of syntactic rules and topological constraints. 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. When users want to find out the causes of an *entity* or an *activity*, their interest is in direct and indirect influences, which can involve multiple transitions to reach the influence’s origin.

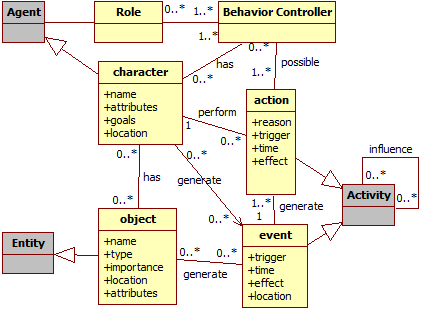
1. Provenance in Games

In order to adopt provenance for the context of games, it is necessary to map each type of vertices from the provenance graph to elements that can be represented in games. As mentioned at section 3, the Open Provenance Model 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 find their counterparts in the game context. To avoid misunderstanding, we adopt throughout this chapter the terms used in PROV (entities, activities, and agents).

Starting with *entities*, their provenance definition states that they are physical or digital objects. This definition already gives a clue about 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 to 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* are defined as 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.

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 to be more suitable to a game context. For example, creating relationships that 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 requires a behavior controller in order to generate and control his actions, providing an array of behavior possibilities. For example, it can be used decision trees [13] to control the NPC’s behaviors. What is required from the behavior controller is to store information when an action is executed. Actions can be represented by a series of attributes that provides 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, what the action produced, and what its affect should be recorded for analysis.



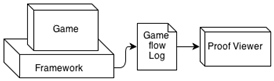
**Fig. .** Data model diagram. Gray classes represent generic provenance classes.

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 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. 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, it is recommended to store relevant data. The way of measuring relevance varies from game to games but ideally it is any information that can be used to aid during analysis process.

1. Provenance Visualization

The purpose of collecting information during a game session is to be able to generate a provenance graph to aid the user to analyze and infer the reasons of the outcome. In this paper we introduce a provenance visualization tool named *Proof Viewer* (Provenance Flow Viewer), which is based on JUNG [14] and allows the analysis of generated game flow log through a graph. A game using the *provenance in games* framework is able to generate *a game flow log* that can be analyzed by *Proof Viewer*. **Fig. 2** illustrates the relationships between the game, using the framework, and *Proof Viewer*.



**Fig. .** Relationships between a game using *provenance in games* framework and the *Proof Viewer*.

First, the *game flow 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 generate the provenance graph. This graph is a representation of the *game flow log* and is available for the user to interact and analyze, reaching his own decisions 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 process. No information is lost in this process, so the user can undo any changes made during analysis.

A small example of a generated provenance graph from exported data is illustrated by **Fig. 3**. Following the provenance notation specification, each vertex shape in **Fig. 3** is related to its type. Square vertex represents *activities* vertices, circles are *entities* vertices and an octagon represents *agent* vertices. The edges in the provenance graph represent relationships between vertices, which can be *agents*, *entities* or *activities*. As such, *activities* vertices can be influenced positively or negatively 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.

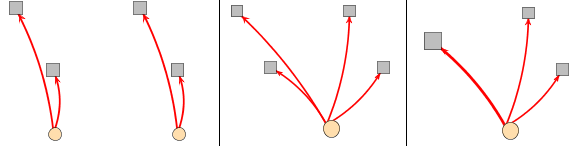


**Fig. .** Example of a generated provenance graph.

*Proof Viewer* has other features besides vertex shape by type. It uses shapes and colors to distinguish displayed information and provides three types of filters: vertex filter, edge filter, and status filter. As previously noted, vertices have different shapes according to their types. However, it is also possible to differentiate a vertex from another with different borders and colors. As an example, *activities* that did not interact with other *activities* are dotted, as illustrated in the upper right corner of **Fig. 3**.

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. It is also possible to make the edge to be dashed in order to emphasize their importance, or lack of. These edge types are also illustrated at **Fig. 3**, 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 this, 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 by **Fig. 4**. The collapsed edge’s information is calculated by the sum or average (depending on the edge type) of the values from the collapsed edges. Another type of filter present is the edge filter, which filters edges by context and by the type of relationship.



**Fig. .** Collapsing vertices. The first picture is the original state showing four *activities* and two *entities* with edges from the same type. The second picture shows the collapse of both *entities* into one. The last picture also shows the collapse of two *activities*, and their respective edges since they were from the same type. Note the size of the resulting edge is bigger than the original ones as a resulting from summing each edge’s values.

The last filter present 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 [15], 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 financial situation throughout the game. When filtered by player’s financial situation, all vertices that contain a player financial value will have their colors changed according to its value. Activating this type of filter allow the user or 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 6 provides more examples of those features.

Using these features for graph manipulation and visualization, the user 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 how the developer planned. It is also possible to hide information that might have not been relevant to the desired analysis. The displayed information can be omitted in the graph or grouped together by features presented in the application.

* 1. Granularity

Depending on the game style, a game session might take several hours to complete, or even days in case of RPGs. This makes the size of the provenance graph to be overwhelming to the user, even when making pre-filtering during the generation of the *game flow 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 use collapses to reduce the graph’s size. Combats can be identified and collapsed into a single vertex for each instance. Places visited in the game can also be collapsed into a single vertex, containing all interactions made in that location, even combats. It is also possible to have collapses inside collapses. In this case, a collapsed combat inside a collapsed area visited by the player may contain other actions aside from the combat, such as interactions with the ambient. This gives an impression of a map from the player’s journey, showing vertices for each location visited by the player, while allowing the player to expand only the situations he desires to analyze. It is similar to *google maps*, where it shows the entire world and allows the user to zoom into specific locations. However in this case, it shows instances of the journey taken by the player.

It is also possible to go beyond that. 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 player. This is possible because provenance is analyzed from the present to the past. This way, combats outcomes are known and can be used to decide if it they are relevant or not. If the player was victorious with minor challenge, did not suffer severe wounds, or barely used any resources at his disposal, then the entire combat can be simplified into just one vertex representing the combat with the enemy. However, if the combat was challenging or the player lost, it is interesting to display all actions in it for analysis, allowing the player to identify important facts that influenced the combat outcome.

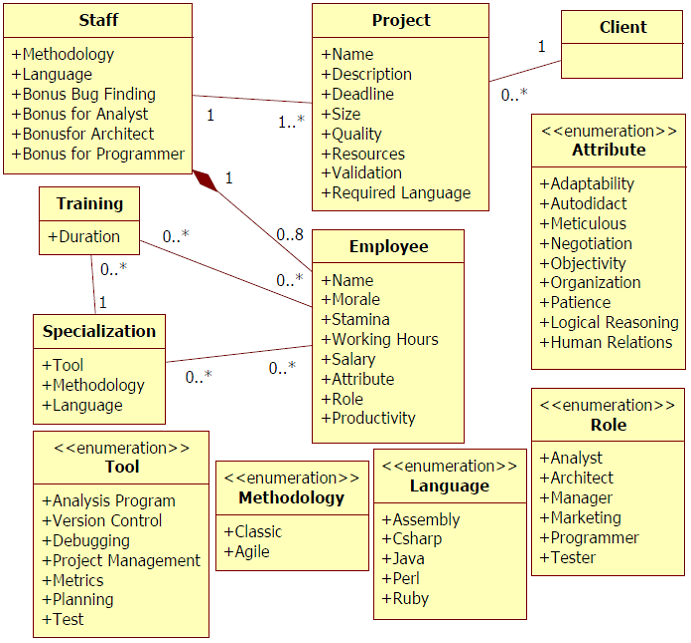
Currently, *Proof Viewer* does not provide inference for the user, only the means necessary to infer. The game developers will need to create their own inference rules customized to their games, such as clustering sequences of actions and identifying irrelevant sections that can be omitted during the graph visualization. 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 involves other areas of research [16–20].

1. Using Provenance in a Game

We instantiated this provenance analysis infrastructure, which uses the proposed framework presented in [1] and described at section 4, in a Software Engineering educational strategy game named SDM (Software Development Manager) [21]. 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 the possibility of tracking mistakes made during a game session or identifying game mechanics that requires tinkering.

In SDM, which was developed using the game engine Unity3D [22], the player has a team of employees that are used to develop software according to contracts made with customers. The gameplay and game mechanics are modeled presenting possibilities to the player to decide strategies for development and define the roles for each staff member. As in any contract, the software has requirements that must be followed during development. From a gameplay point of view, these requirements help to balance the mechanics and rules. When the software is completed and delivered to the customer, there is a quality assessment of the software and a project completion payment accordingly to the product quality.

Since SDM focuses in people management, the main elements of the game are the employees, which represent the player’s labor force. Employees can perform different roles (analyst, architect, manager, marketing, programmer, and tester.), which uses the employee’s attributes to calculate his performance depending on the role performed. Another element present in the game is specialization, used to define the employee working competence. With the specialization system, it is possible for employees to undergo training to learn new sets of skills. Also the concepts of working hours, morale, and stamina are used to modify the employee’s productivity. **Fig. 5** shows a simplified version of SDM’s class diagram focusing on the employee, showing his human attributes, types of specializations, the possibility of training to acquire specializations, and that the employee is affect by other employees that belong to the staff team. It also illustrates the project, its characteristics and requirement.



**Fig. .** SDM simplified class diagram

* 1. Information Storage

The information structure used on SDM is similar to the one explained in [1]. As such, each project contains a list of employees that were 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 for 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 [1] and illustrated by **Fig. 1**. Each vertex contains different information according to its type.

*Activities* vertices, which represent actions executed during the game by employees, store information about its execution. 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 and credits spent to execute the action are also stored. Lastly, the progress made in his task and a description of the action, explaining his decision making process. Besides those, if the action had any external influences, or used or altered an artifact, a link to the action that affects its execution and the artifact is included.

*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. After the data is collected and extracted, a provenance graph corresponding to that scenario is generated and displayed for analysis, similar to the one presented by **Fig. 3**.

* 1. Provenance Graph

With the adaptations made in the original SDM [1], it is now possible to collect data and use it to generate a provenance graph. The collected game data, known as *game flow log*, is exported to *Proof 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]](#footnote-1), **Fig. 6** 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 resignations in brown.

**Fig. 6** was already subject to 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 for an *agent* and thick red edge for 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* A.



**Fig. .** An example of credits status filter.

In vertex 3, the player’s credits changed to red zone due to payments, meaning that his resources are almost empty and will not have enough credits to keep paying his employees. When that happens, employee’s 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. 7**, we can see 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.



**Fig. .** Non-collapsed graph from **Fig. 6** using filter: Morale

This analysis can be used to detect player’s behaviors and the reasons of why they are losing the game. In the example, the cause was lack of resources due to hiring a new employee. Was it necessary to hire a new employee in order to finish the game? If so, then there is a problem that requires immediate attention since the game requires the player to hire a new employee in order to complete his objective. However, hiring an employee instead causes the player to lose the game. 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.



**Fig. .** Example of a provenance graph analysis.

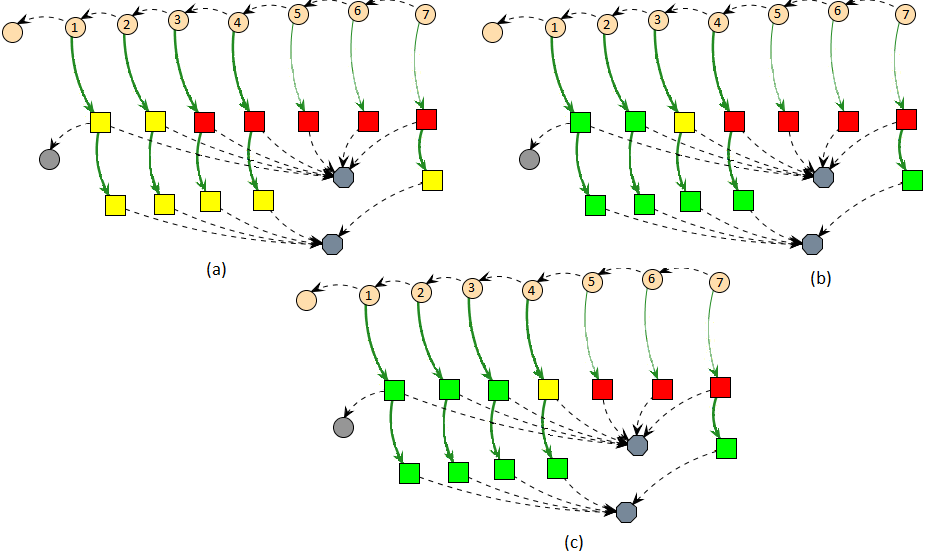
Another example of analysis is by checking employee productivity and understanding why variations occurred by using multiple filters to test theories. In SDM, productivity is defined by the executed task, the amount of outside help, the employee’s job (junior, mid-level, and senior), the working hours, and the stamina and morale stats. **Fig. 8** illustrates an example scenario. To simplify the graph visualization due to size limits, we focus only on two *agents* and the main *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 did an ad hoc approach, which maximizes his productivity at the cost of quality. The change in vertex 3 can be identified by looking at his working hours, which can be done by looking at each individual vertex or by adding a filter, as shown in **Fig. 9**.

In **Fig. 9** we 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 in **Fig. 9** that in vertex 3 his 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 is due to 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.



**Fig. .** Graph from **Fig. 8** using filters: working hours (a), stamina (b), and morale (c).

1. Conclusion

This paper introduces new perspectives on software engineering learning, leveraging the current state of the art, based on gameplay, to a level where the game provenance can produce and consolidate knowledge. This knowledge can help on (1) confirming the hypotheses formulated by students, (2) supporting tutors for a better guidance, (3) motivating group dynamics around some case studies, and (4) extracting behavior patterns from individual sessions or groups of sessions.

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

Currently, we do not make inferences to the user, but let the user or developers to 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 flow. Lastly, we are working on different graph visualization layouts and also running experimental studies on the usage of provenance in educational games to evaluate the aspects of learnability.

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

References

1. Kohwalter, T., Clua, E., Murta, L.: Provenance in Games. 2012 XI Brazilian Symposium on Games and Digital Entertainment (SBGAMES). In: XI SBGames, Brasilia (2012).

2. Freire, J., Koop, D., Santos, E., Silva, C.T.: Provenance for Computational Tasks: A Survey. Computing in Science Engineering. 10, 11 –21 (2008).

3. Warren, C.: Game Analysis Using Resource-Infrastructure-Action Flow, http://ficial.wordpress.com/2011/10/23/game-analysis-using-resource-infrastructure-action-flow/.

4. Consalvo, Mi., Dutton, N.: Game analysis: Developing a methodological toolkit for the qualitative study of games. Game Studies. 6, (2006).

5. Andersen, E., Liu, Y.-E., Apter, E., Boucher-Genesse, F., Popović, Z.: Gameplay analysis through state projection. Proceedings of the Fifth International Conference on the Foundations of Digital Games. pp. 1–8. ACM, New York, NY, USA (2010).

6. Cavazza, M., Charles, F., Mead, S.J.: Character-based interactive storytelling. IEEE Intelligent Systems. 17, 17 – 24 (2002).

7. Passos, E.B., Montenegro, A.A., Clua, E.G., Pozzer, C.T., da Silva, F.S.C.: Hierarchical PNF Networks - A Temporal Model of Events for the Representation and Dramatization of Storytelling. 2009 VIII Brazilian Symposium on Games and Digital Entertainment (SBGAMES). pp. 175 –184 (2009).

8. Pinhanez, C.S., Bobick, A.F.: Human action detection using PNF propagation of temporal constraints. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1998. Proceedings. pp. 898 –904 (1998).

9. Moreau, L., Foster, I., Freire, J., Frew, J., Groth, P., McGuiness, D.: IPAW, http://www.ipaw.info/.

10. 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). In: Future Generation Computer Systems. 27, 743–756 (2007).

11. Miles, S., Heasley, J., Szalay, A., Moreau, L., Groth, P.: Provenance Challenge WIKI, http://twiki.ipaw.info/bin/view/Challenge/.

12. Moreau, L., Missier, P.: PROV-DM: The PROV Data Model, http://www.w3.org/TR/prov-dm/.

13. Moret, B.: Decision Trees and Diagrams. In: ACM Computing Surveys (CSUR). 14, 593–623 (1982).

14. Joshua O’Madadhain, Danyel Fisher, Tom Nelson: JUNG: Java Universal Network/Graph Framework. Open-source, sourceforge (2010).

15. Diehl, S.: Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer (2007).

16. Bristol, E.H.: Pattern recognition: An alternative to parameter identification in adaptive control. Automatica. 13, 197–202 (1977).

17. Cios, K., Pedrycz, W., Swiniarski, R.W.: Data mining methods for knowledge discovery. Kluwer Academic Publishers, Norwell, MA, USA (1998).

18. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P.: From Data Mining to Knowledge Discovery in Databases. AI Magazine. 17, 37 (1996).

19. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2006).

20. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, Second Edition. Morgan Kaufmann (2005).

21. Kohwalter, T., Clua, E., Murta, L.: SDM – An Educational Game for Software Engineering. 2011 X Brazilian Symposium on Games and Digital Entertainment (SBGAMES). In: X SBGames, Salvador (2011).

22. Higgins, T.: Unity - 3D Game Engine, http://unity3d.com/.

1. In order to reduce graph size and provide a quicker understanding for the examples presented, some in game parameters were set to allow faster state transitions. [↑](#footnote-ref-1)