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

Troy C. Kohwalter

Esteban G. W. Clua

Leonardo G. P. Murta

Universidade Federal Fluminense  
Instituto de Computação, Passo da Pátria, 156, bloco E, Niterói - RJ Brasil

{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, used for analysis. As a proof of concept, we also instantiated our proposed framework and graph generation in a Software Engineering game, allowing players to identify their mistakes and learn through them by analyzing the generated provenance graph from collected gameplay data.

**Categories and Subject Descriptors**

I.2.1 [**Applications and Expert System**]: Games; K.3.0 [**Computer Uses in Education**]:General; K.8.0 [**Personal Computing**]: General - Games

**General Terms**

Documentation, Design, Human Factors, Theory.

**Keywords**

Provenance, education, game analysis, action flow, graph, storytelling, learning.

# INTRODUCTION

The conclusion of a game session derives from a series of decisions and actions made during the game. In many situations, analyzing and understanding the events, mistakes, and flows of a concrete game play may be useful for understanding the achieved results. Traditional games are limited in terms of analysis of the obtained results and do not allow the player to comprehend the story development throughout the game.

In many cases, the game flow analysis is fundamental for detecting symptoms of problems that occurred due to wrong decision-making or even bad gameplay project. Without it, 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 [1], 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 [4, 20] due to their stimulating curiosity characteristic and for providing motivation for learning [24]. 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.

Neural studies about the learning capability of human brain [7, 9] 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.

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 in the outcome. In order to improve understanding, we provide the means to analyze the game flow by using provenance. Provenance refers to the documented history of an object's life cycle and is generally used in the context of art, digital data, and science [23].

In our previous work [16], we introduced the usage of digital provenance [12] 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.

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 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 this work, the provenance in games framework is instantiated in the SDM game [17] as a proof of concept. The SDM game focuses on introducing Software Engineering concepts and skills to undergraduate students. The new and improved version of SDM presented in this paper includes provenance gathering and analysis, allowing students to visualize their actions and identify steps that lead to successful or unsuccessful outcomes. While the main application of provenance 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 some background on provenance, explaining key definitions that are used in the provenance graph, as well as the framework used to enable provenance in games. Section 4 presents the provenance analysis by using a provenance graph and some features to aid in the analysis. Section 5 presents a proof of concept usage of the analysis in the SDM game. Finally, Section 6 presents the conclusions of this work and points out some future work.

# RELATED WORK

Warren [13] proposes an informal method to analyze the game flow using a flow graph, mapping game actions and resources to vertexes. 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 [10] 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 [2], offers a way to visually analyze play steps, providing detailed visual representation of the actions taken by the player through the game.

Besides [13], 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 [21]. This method organize the story using PNF networks [22], 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.

# 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)*,* the participants were interested in the issues of data provenance, documentation, derivation, and annotation. As a result, the *Open Provenance Model* (OPM) [18] was created at the *Provenance Challenge*, which is a collocated event of IPAW. Recently, another provenance model was developed, named PROV [25], 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 [18], 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 nodes that can represent *Artifacts*, *Processes*, and *Agents*. *Artifacts* are an immutable entity that can represent a physical object or its digital representation in a computer system. *Processes* are actions or a sequence of actions performed over artifacts and producing new artifacts. *Agents* are contextual entities acting as a catalyst of a process that can enable, facilitate, control, or affect its execution. The edges of the graph belong to one of the categories described in Figure 1, representing a causal dependency between the source, which denotes the effect, and the destination, which denotes the cause. Following we present some important definitions introduced by OPM [18].

**Artifact Used by a Process**: A *used* edge from process to an artifact is a causal relationship intended to indicate that the process required the availability of the artifact to be able to complete its execution. When several artifacts are connected to the same process by multiple *used* edges, all of them were required for the process to complete.

**Artifacts Generated by Processes**: A *was generated by* edge from an artifact to a process is a causal relationship intended to mean that the process was required to initiate its execution in order to generate the artifact. When several artifacts are connected to the same process by multiple *was generated by* edges, the process was responsible to produce all of them.

**Process Triggered by Process**: An edge *was triggered by* from a process P2 to a process P1 is a causal dependency indicating that the start of process P1 was required to allow P2 to complete.

**Artifact Derived from Artifact**: An edge *was derived from* from artifact A2 to artifact A1 is a causal relationship indicating that artifact A1 should have been generated in order to generate A2. The piece of state associated with A2 is dependent on the presence of A1 or on the piece of state associated with A1.

**Process Controlled by Agent**: An edge *was controlled by* from a process P to an agent Ag is a causal dependency indicating that agent Ag controlled the start and end of process P.

# 

Figure 1. Example of edges and timestamps in OPM. Source: [18].

In Figure 1, the edge *used* means that a process used an artifact, while *was generated by* edge indicates that an artifact was generated by a process. The letter "R" represents the roles under which these artifacts were used since a process may have used several artifacts. Likewise, a process may have generated many artifacts, and each would have a specific role. Roles are only meaningful in the context of the process where they are defined, and they are not defined by the OPM itself, but by the application domains. Roles are used on OPM to distinguish the involvement of artifacts in processes.

The edge *was controlled by* means that an agent managed the process, essentially acting as a catalyst or controller. Since several agents may have controlled a process, their roles are also identified as controllers. This type of dependency represents a control relationship and not a data derivation. The edge *was derived from* assert that artifact A2 was derived from another artifact A1, giving a dataflow view of the provenance. In contrast to the edge *was derived from*, an edge *was triggered by* provides a control flow view of the provenance. Figure 2 illustrates the provenance class diagram with edges types.

Moreover, the provenance model allows causality graphs to be used with time information. In this model, time is not used for deriving causality, but to validate causality claims, since if the same time clock is used to measure the time for both the effect and cause, then the time of an effect should be greater than the time of its cause.

In addition, time may be associated to *instantaneous occurrences* in a process. There are four types of occurrences, being denoted as *creation* and *use* for artifacts and *starting* and *ending* for processes. Given that time may be observed by someone, its accuracy is limited by the clock and the notion of time. This way, the model considers an interval of accuracy to support the granularity used to represent time. With this, it is possible to state that an artifact was used no earlier than time t1 and no later than time t2, as an example. This rationale is analogous for processes.

Figure 1 indicates how time information can be expressed in the model. For *used* and *was generated by* edges, one timestamp can be used to express when the event happened. For *was controlled by* edge two timestamps mark when the process started and terminated. For *was derived from* and *was triggered by* edges, one timestamp is adopted to indicate when the artifact was used. Despite using timestamp, the time of occurrence itself is not enough to imply causality. The fact that process P1 happened before P2 is not enough information to infer that P1 triggered P2.

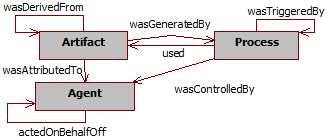


Figure 2. Provenance Class Diagram

Finally, the Open Provenance Model has defined the notion of a graph based on a set of syntactic rules and topological constraints. The provenance graph captures causal dependencies that can be summarized by means of transitive closure. Because of this, a set of completion rules and inferences can be used in the graph. When users want to find out the causes of an artifact or a process, their interest is in indirect causes that involve multiple transitions.

## Provenance in Games

The mapping of provenance nodes to their game counterpart is necessary to use a provenance graph for game flow analysis. We first proposed in [16] an adoption of provenance in the context of games, mapping each provenance node type to elements typically present in games. In such mapping, *Artifacts*, which consist on "*an immutable piece of state that can represent a physical object*" [18], were mapped to game objects. This way, artifacts can be anything used in the game, such as weapons, potions, legendary artifacts, magical objects, among others, when considering a typical RPG game. In essence, it can represent anything meaningful to the development of the game story or to the scenery.

A*gents*, which "*are contextual entities acting as a catalyst of a process that can enable, facilitate, control or affect its execution*" [18], were mapped as characters present in the game, such as non-playable characters (NPCs), players, and other entities, which can also be plot-managing entities.

Lastly, *Processes*, which are "*actions or a sequence of actions performed or caused by artifacts*" [18], were mapped to actions or events made by entities in the game. In [16], the generation of actions and events are controlled by decision trees [19]. However, any decision making algorithm can be used instead to control actions and behaviors. These generated actions and events are represented as *processes* nodes in the provenance graph.



Figure 3. Data model diagram. Gray classes represent provenance classes. Source: [16].

Moreover, in [16] we proposed a data model for provenance in games, which illustrates the provenance mapping and information examples that can be used for analysis, as shown by Figure 3. Besides this, [16] also presented an information structure to store collected game data to generate a game flow log for provenance analysis.

# PROVENANCE ANALYSIS

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 this paper we introduce a novel provenance visualization tool named *Proof Viewer* (Provenance Flow Viewer), which is based on JUNG [15] 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*. Figure 4 illustrates the relationships between the game, using the framework, and 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 nodes following our defined 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 the outcome. The user is able to see the consequences of each action and how they influenced other actions and the outcome, as well as 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 player can undo changes made during analysis.

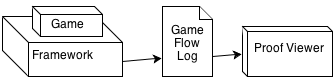


Figure 4. Relationships between a game using provenance in games framework and *Proof Viewer*.

A small example of a generated provenance graph from exported data is illustrated by Figure 5. Figure 5Figure 1 As can be seen in the figure, there is a chain of artifact nodes that represents the graph’s backbone, The edges in the provenance graph represent relationships between nodes, which can be *agents*, *artifacts* or *processes*. As such, *processes* nodes can be influenced positively or negatively by other *process* and have relationships with *artifacts* and *agents*. The context of such relationships may vary according to the type of relation between nodes.

Proof Viewer has other features besides node shape by type. It uses shapes and colors to distinguish displayed information and provides three types of filters: node filter, edge filter, and status filter. As previously noted, nodes have different shapes according to their types. However, it is also possible to differentiate a node from another with different borders and colors. As an example, *processes* that did not interact with other *processes* can be dotted, as illustrated in Figure 5. It is also possible to use different formats for edges. The thickness can be interpreted as how strong the relationship is. If the edge represents a low influence on the *process*, it is drawn as a thin edge. If the influence is high, then it becomes a thicker edge.

Another resource present for edges is color to represent the type of relationship. There are three types of relationship: positive, which indicates a beneficial relation; negative, which is a benign relation; and neutral, which is neither beneficial nor benign. 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. To emphasize the neutral relationships lack of importance, they can also be dotted. These edge types are illustrated in Figure 5.

In order to better analyze graph data, the node filter feature is also available. Since the graph is generated from collected game date, 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 nodes in order to reduce the graph size by changing the information display scale, grouping nearby nodes together and thus changing the graph granularity. Another usage of collapse is to group *processes* from the same *agent*, making easier to see all influences and changes that the *agent* did throughout the game. Figure 6 illustrates a collapse by grouping an *agent’s processes* with the *agent*. Note the node size difference when collapsed. Another type of filter present is the edge filter. In the application it is also possible to filter edges by context and by the type of relationship.



Figure 5. Example of a generated provenance graph.

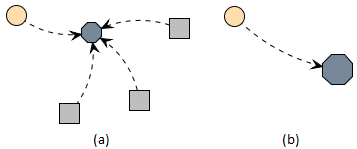


Figure 6. Collapsing Nodes: the normal graph, containing an artifact, an agent, and three processes (a) and the grouping of all agent processes with the agent (b).

The last filter present is the status filter. When selecting the desired attribute, all nodes with the specified status will have their colors changed according to their respective values. It uses the traffic light scale, 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 nodes that contain a player HP value will have their colors changed according to its value. Activating this type of filter allow the user to see the player’s HP throughout the game, making it easier to identify situations where he might have had trouble (red color). Section 5 provides more examples of those features used for analysis.

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 is also possible to hide information that might not be relevant to the desired analysis. Irrelevant information can be omitted in the graph or grouped together by features presented in the application.

As an example, suppose the player is in combat with an enemy and only after a few rounds it falls under the player's attacks. With the framework proposed in [16], every interaction creates a node to represent the action taken by the player, which is attacking the enemy. This may generate data that is unnecessary for analysis, so it is possible to reduce all the individual attack nodes to simply one node. Another case could consist in a combat that does not generate any impact in the story outcome. In this case, it could be completely omitted.

However, the player could have made other actions against the enemy, which are also considered as forms of attack, such as casting a spell, a special attack maneuver, or even healing himself in order to survive. These actions are not duplicated, but can still be encapsulated for a general analysis, and, if necessary, expanded for a detailed analysis. Note that all collected information is preserved and the only change made is on how it is displayed.

Since provenance is analyzed from the present to the past, 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 node representing that the player attacked the enemy and was victorious. However, if the combat was challenging or the player lost, it is interesting to show all action nodes for analysis, allowing the player to identify important facts that influenced the combat outcome.

Note however that Proof Viewer does not provide inference for the user, only the means necessary to infer. The player himself will need to decide which information is relevant for analysis. 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 [5, 8, 11, 14, 27].

# USAGE EXAMPLE

We instantiated this provenance analysis infrastructure, which uses the proposed framework presented in [16], 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. As so, the adoption of provenance becomes an important instrument to better support knowledge acquisition, allowing the possibility of tracking mistakes made during a game session.



Figure 7. SDM's simplified class diagram.

In SDM, which was developed using the game engine Unity3D [26] and had some elements influenced by [3], 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.

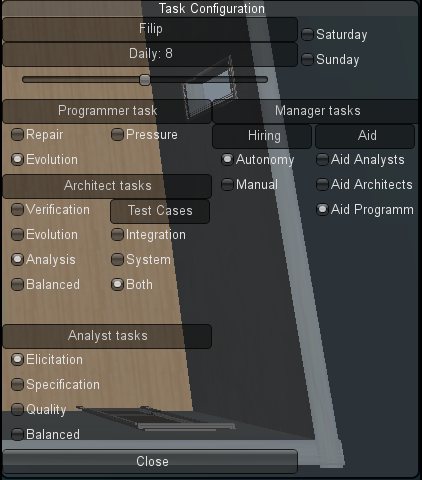


Figure . Task Configuration window.

Since SDM focuses in people management, the main elements of the game are employees, which represent the player’s labor force. Employees can perform different roles (analyst, architect, manager, marketing, programmer, and tester), which uses employee’s attributes to calculate his performance. 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. Figure 7 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 how an employee is affect by other employees that belong to the staff team. It also illustrates the project, its characteristics and requirements. We included in [16] different changes in the game, introducing new concepts and expanding employee behaviors and actions, including some significant role responsibilities changes. Figure 8 illustrates new tasks available for each role. Programmers are now responsible for the software evolution and repair. Architects create prototypes to aid the analyst during elicitation as well as generate test cases. Analysts write specification and generate test cases. Lastly, the manager had his aiding task divided into three different types.

## Information Storage

The information structure used on SDM is similar to the one explained in [16]. 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, when actions are executed, information is collected and stored for generating the provenance graph used during post-game analysis. Since provenance graphs contains three types of nodes (*processes*, *agents*, and *artifacts*), the collected information is mapped to each type, according to the data model explained in [16] and illustrated by Figure 3. Each node contains different information according to its type.

*Processes* nodes, which represents actions executed during the game by employees, stores 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* nodes, representing employees, store the employee’s name, his current staff grade, his level, human attributes which are used in the game, and specializations.

*Artifact* nodes vary according to its type. In SDM we use three types of *artifacts*: Prototypes, produced by architects and consumed by analysts; Test Cases, produced by analysts, architects, and programmers and consumed by testers; and Project, which represents the software development progress and is considered the main *artifact* for analysis. The project type *artifact* stores project information such as, the day of its instance, the project’s deadline, how much coding was produced and the code overall quality. It also stores the clients requirements identified and modeled by analysts, how many credits is paid each month to the player, how much credits the player has available, and the state of each type of bugs in the software. Figure 9 illustrates an example of the data shown in Proof Viewer according to the node’s type.

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 Figure 5.

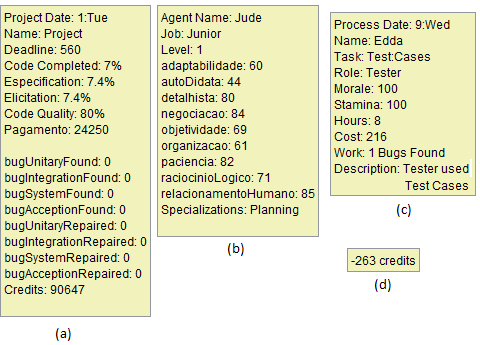


Figure 9. Information data in the provenance graph. The project’s *artifact* node data (a), employee’s *agent* node data (b), action’s *process* node data (c), and an edge data (d).

## Provenance Graph

With the adaptations made in the original SDM concepts [16], it is possible to use the collected data for provenance analysis. The collected game data is exported to *Proof Viewer*. In that application, the data is processed and used to generate a provenance graph for analysis.



Figure 10. An example of credits status filter.



Figure 11. Non-collapsed graph from Figure 10 using filter: Morale.

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), Figure 10 illustrates a scenario where the player had credits problem. Credits are the monetary coin in the game. To simplify the picture, some collapses were made, omitting most of the *agent’s* *processes*. The *artifacts* represent instances of the development stage, including the player’s credit value. The *processes* present in the picture represent hiring actions in gray and resignations in brown.

Figure 10 was already subject to credits filter, both in the edges and in the nodes. In node 1, the project had a substantial credits 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 *process*. The player’s credits are also in a green zone as marked by the project’s node color. However, due to the hiring fee paid in node 1 and the resources used by the staff in node 2, the player’s credits changed to a yellow zone, even with the minor income from *agent* A. In node 3, the player’s credits changed to red zone due to payments process, 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 *processes*. Observing Figure 11, 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 looses the game.

Another example of analysis is by checking employee productivity and understanding why variations occurred. 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. Figure 12 illustrates an example scenario. To simplify the graph visualization due to size limits, we focus only in two *agents* and the main *artifact* known as “project”. Those *agent’s* 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 nodes 1 to 7. We can also see that the manager did not cause this fluctuation, since his aid bonus did not have much variation. The variation from nodes 1 to 2 in the programmer’s productivity was because the programmer performed different tasks. There are three different types of tasks in the picture for the programmer: Test-driven, which has reduced productivity for more quality, design-code, which is treated as the default type of programming, and ad hoc, which increases productivity at the cost of quality. By analyzing node details illustrated at Figure 13, the programmer did a test-driven approach in node 1, which reduces his productivity.



Figure 12. Example of a provenance graph analysis.

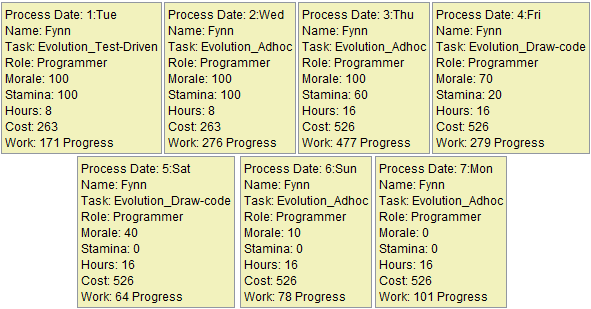


Figure 13. Programmer’s node details from Figure 12.

In node 2 he did an ad hoc approach, which maximizes his productivity at the cost of quality. The change in node 3 can be identified by looking at his working hours, which can be done by looking at each individual node or by adding a filter, as shown in Figure 14.

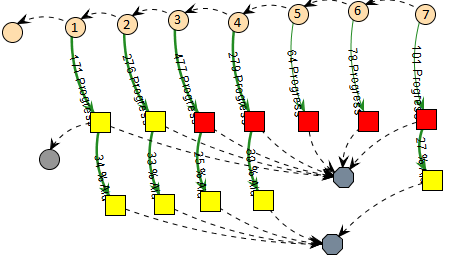
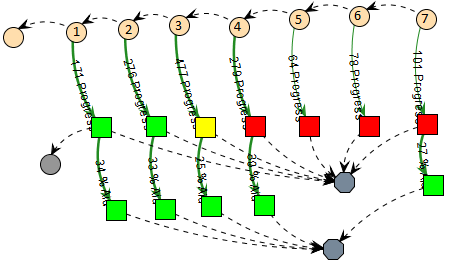


Figure 14. Graph from Figure 12 using filter: working hours.



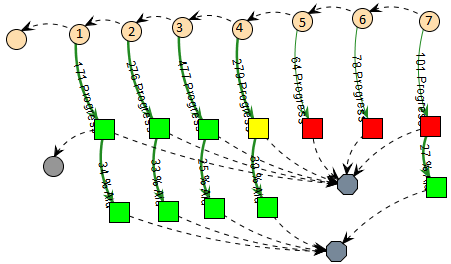
**Figure 15. Graph from Figure 12 using filter: stamina.** 

Figure 16. Graph from Figure 12 using filter: morale.

In Figure 14 we can see via the change from yellow to red that the programmer’s working hours per day increased. Since the *process* node in node 3 is red, it means the employee is doing extra hours, which increases his productivity. From nodes 3 to 7, his working hours remained unaltered. Therefore, the change from nodes 2 to 3 was mainly due the change on his daily working time. However, if we look at node 4, we can see a drop in his productivity.

The task performed at node 4 was design-code, as showed by Figure 13, but even so there was a subtle decrease due to changes in his stamina and morale. In node 5 the change is more visible, especially because he was in ad hoc mode, same as node 3. His working hours had not decreased, as noted in Figure 14. This drastic change probably occurred because the programmer was getting tired, meaning his stamina was dropping because of the extra hours he was doing.

By changing the filter again to show stamina levels, we can see in Figure 15 that in node 3 his stamina indeed dropped to yellow because of the extra hours and in node 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 node 5, as illustrated by Figure 16. Lastly, the small variation from nodes 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.

# CONCLUSION

This paper proposed a new type of game flow analysis by using a provenance in games framework. It allows post game analysis to discover 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 the player 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.

In this paper we presented the Proof Viewer, a tool to visually aid the analysis by manipulating the generated provenance graph from collected data. We also showed a game in which our proposed provenance analysis was instantiated, with some flow and elements examples. During the game session, information is collected, generating a game flow log, which is used in a post analysis to generate a provenance graph. We also explained some analysis features that can be done in the provenance graph to aid or refine the analysis.

However, there are still some limitations. If the provenance graph is too large, by the order of thousands of nodes, even using the granularity filter, there will be some visualization limitations, not to mention performance degradation when executing filters. Due to the graph size, information contained in collapsed nodes will also have visualization issues. This happens because the collapsed node is actually a graph, and the information displayed is a list of all nodes contained in it with their respective information. One way to deal with this problem is to filter the displayed information from the graph-node to only a few, more important or relevant ones, and display the visualization of the collapsed graph in a separated window.

Currently, we do not make inferences to the user, but let the user decide what he wants to infer. Studies in this area can be made in order to identify information that can be safely omitted from the user without affecting the overall analysis. Another interesting research is to automatically identify patterns in the game flow. Lastly, we plan on working in different graph visualization layouts and run experiments to evaluate the aspects of learnability of using the provenance graph in order to understand better the story.

# ACKNOWLEDGMENTS

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

# REFERENCES

[1] Abt, C.C. 1987. *Serious Games*. University Press of America.

[2] Andersen, E. et al. 2010. Gameplay analysis through state projection. *Proceedings of the Fifth International Conference on the Foundations of Digital Games* (New York, NY, USA, 2010), 1–8.

[3] App Store - Game Dev Story: 2010. *http://itunes.apple.com/us/app/game-dev-story/id396085661?mt=8*. Accessed: 2011-05-05.

[4] Baker, A. et al. 2003. Problems and Programmers: An Educational Software Engineering Card Game. *International Conference on Software Engineering(ICSE)*. (2003), 614–621.

[5] Bristol, E.H. 1977. Pattern recognition: An alternative to parameter identification in adaptive control. *Automatica*. 13, 2 (Mar. 1977), 197–202.

[6] Cavazza, M. et al. 2002. Character-based interactive storytelling. *IEEE Intelligent Systems*. 17, 4 (Aug. 2002), 17 – 24.

[7] Chialvo, D.R. and Bak, P. 1999. Learning from mistakes. *Neuroscience*. v. 90, 4 (Jun. 1999), 1137–1148.

[8] Cios, K. et al. 1998. *Data mining methods for knowledge discovery*. Kluwer Academic Publishers.

[9] Clark, G. 1950. The organization of behavior: A neuropsychological theory. *The Journal of Comparative Neurology*. v. 93, 3 (1950), 459–460.

[10] Consalvo, Mi. and Dutton, N. 2006. Game analysis: Developing a methodological toolkit for the qualitative study of games. *Game Studies*. 6, 1 (Dec. 2006).

[11] Fayyad, U. et al. 1996. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*. 17, 3 (Mar. 1996), 37.

[12] Freire, J. et al. 2008. Provenance for Computational Tasks: A Survey. *Computing in Science Engineering*. 10, 3 (Jun. 2008), 11 –21.

[13] Game Analysis Using Resource-Infrastructure-Action Flow: 2011. *http://ficial.wordpress.com/2011/10/23/game-analysis-using-resource-infrastructure-action-flow/*. Accessed: 2012-10-19.

[14] Han, J. and Kamber, M. 2006. *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

[15] Joshua O’Madadhain et al. 2010. *JUNG: Java Universal Network/Graph Framework*. Open-source.

[16] Kohwalter, T. et al. 2012. Provenance in Games. *2012 XI Brazilian Symposium on Games and Digital Entertainment (SBGAMES)* (Brasilia, Nov. 2012).

[17] Kohwalter, T. et al. 2011. SDM – An Educational Game for Software Engineering. *2011 X Brazilian Symposium on Games and Digital Entertainment (SBGAMES)* (Salvador, Nov. 2011).

[18] Moreau, L. et al. 2011. The Open Provenance Model core specification (v1.1). *In: Future Generation Computer Systems*. 27, 6 (Jun. 2011), 743–756.

[19] Moret, B. 1982. Decision Trees and Diagrams. *In: ACM Computing Surveys (CSUR)*. 14, 4 (Dec. 1982), 593–623.

[20] Navarro, E. 2006. *Simse: a software engineering simulation environment for software process education*. California State University at Long Beach.

[21] Passos, E.B. et al. 2009. 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)* (Oct. 2009), 175 –184.

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

[23] PREMIS Working Group 2005. *Data Dictionary for Preservation Metadata*. Technical Report #Final Report of the PREMIS Working Group. Implementation Strategies (PREMIS).

[24] Prensky, M. 2001. Fun, Play and Games: What Makes Games Engaging. *Digital Game-Based Learning*. (2001), 1–31.

[25] PROV-DM: The PROV Data Model: 2012. *http://www.w3.org/TR/prov-dm/*.

[26] Unity - 3D Game Engine: 2010. *http://unity3d.com/*. Accessed: 2011-05-05.

[27] Witten, I.H. and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann.

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