Provenance in Games

Authors omitted for Blind Review

Institution omitted for Blind Review

Abstract

Serious games have been used to aid the understanding of concepts that are taught in theoretical classes. However, mistakes made by players may result in failure to complete the game objectives. These mistakes, which are usually difficult to spot or reproduce in subsequent trials, directly jeopardize the learning capabilities of the serious games and are usually avoided by the established gameplay. In order to solve this issue, this paper introduces a new concept: provenience in games. To do so, we present a framework that records all useful gameplay data and provides this data to further analysis. We also instantiated this framework in a Software Engineering game as a proof of concept, which allows the player to identify his mistakes and learn through them.

Keywords: serious games, education, game analysis, provenance, action flow.

Authors' contact:

Omitted for Blind Review

1 Introduction

Games have been used for aiding students to learn and comprehend concepts taught in classrooms [Navarro and van der Hoek 2004; Baker et al. 2003; Dantas et al. 2004; Figueiredo et al. 2010]. However, traditional serious games are limited in terms of analysis, and do not allow the player to deeply comprehend decisions made throughout the game. In many cases, this analysis is fundamental for detecting symptoms of problems that occurred due to wrong decision-making. The player would be required to play the game again and make different decisions to intuitively figure out which ones were not adequate to the situation. However, depending on the dynamics and the complexity of the game, reproducing the same state can be unviable, making it difficult to replay it and try new solutions.

Neural studies about the learning capability of human brain [Chialvo and Bak 1999; Clark 1950] state that the process of learning by correcting past mistakes can be very efficient. This process increases the 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 are the mistakes.

A method to analyze the game flow using a flow graph, which maps actions, was informally proposed by WARREN [2011]. More formal approaches were also proposed by [Consalvo and Dutton 2006], in which the analysis is done by metrics collected during the game session, creating a gameplay log to identify events caused by player choices. Another method, called *Playtracer* [Andersen et al. 2010], offers a way to visually analyze play steps, providing detailed visual representation of the actions taken by the player through the game. Besides WARREN [2011] proposal, which is superficially described in a blog, the other two methods are developer-oriented, meaning 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 in the context of serious games.

The goal of this paper is to introduce the use of provenance to better support learning in the context of serious games. Our proposal is composed of a framework, which collects the necessary information from the game session while it is being played for provenance usage. This collected data, in a future work, will be processed to create an oriented graph, which maps the actions flow made during the game session and the generated outcomes. This graph is then visible to the player, allowing him to analyze and identify critical nodes that influenced the game outcome. Doing so, it allows him to understand how the outcome was obtained and the decisions that influenced it. This also aid in the identification of mistakes, allowing the player to reflect upon them for future interactions.

Our framework was instantiated in the SDM game [Omitted for blind review] as a proof of concept. The SDM game focuses on introducing Software Engineering concepts and skills to undergraduate students. The new version of SDM, which includes provenance support, allows students to analyze their actions and clearly identify steps that lead to successful or unsuccessful outcomes. This scenario is especially representative because there are multiples influences that may lead to success or failure in a software project.

This paper is organized as follows: Section 2 provides some background on the Open Provenance Model, explaining some of key definitions that are used by the proposed method in order to create the action graph. Section 3 presents the proposed framework to integrate provenance into games, explaining how the structure is organized and giving some examples to the game-provenance mapping. Section 4 presents a proof of concept usage of the proposed framework on the SDM game, pointing to the changes made in order to adapt it to support provenance. Finally, Section 6 presents the conclusions of this work and points out some future work.

2 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*, the participants were interested in the issues of data provenance, documentation, derivation and annotation. As a result, the Open Provenance Model (OPM) [Moreau et al. 2011] was created from the Provenance Challenge that was held in that workshop.

The Open Provenance Model is a proposed model of provenance that was designed to meet the following requirements [Moreau et al. 2011]:

1. Allow provenance information to be exchanged between systems;
2. Allow developers to build and share tools to operate on such provenance model;
3. Define provenance in a precise, technology-agnostic manner;
4. Support digital representation of provenance;
5. Allow multiple levels of description to coexist;
6. Define a core set of rules that identify the valid inferences that can be made on provenance representation.

In Open Provenance Model, it is assumed that provenance of objects are represented by an annotated causality graph, which is a directed acyclic graph enriched with annotations capturing further information pertaining to execution. According to MOREAU *et al.* [2011], a provenance graph is a record of a past or current execution, and not a description of something that could happen in the future.

The causality graph is composed of nodes that can represent *Artifacts*, *Processes* and *Agents*. *Artifacts* are an immutable piece of state that can represent a physical object or a digital representation in a computer system. *Processes* are actions or a sequence of actions performed or caused by artifacts and results

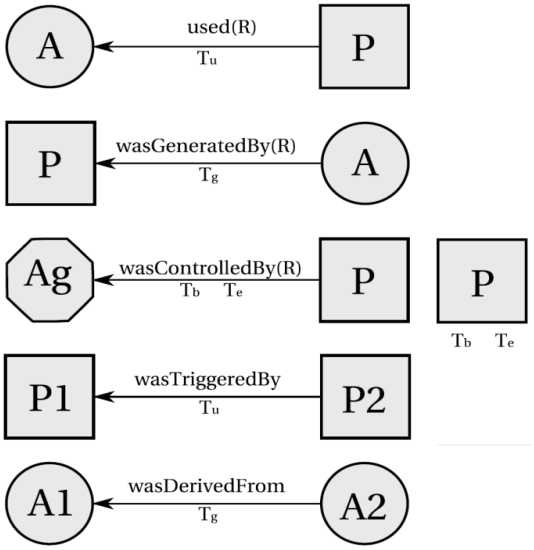

Figure 1: Edges and Usage of Timestamps in OPM. Source: [Moreau et al. 2011].

in 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 its source, denoting the effect, and its destination that denotes the cause. Below are some important definitions in the Open Provenance Model according to MOREAU *et al.* [2011].

Causal Relationship: Represented by an arc and denotes the presence of a causal dependency between the source (effect) and the destination (cause).

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 a 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 must begin for all of them to be generated.

Process Triggered by Process: An edge [*was triggered by*] from a process P2 to a process P1 is a causal dependency that indicates that the start of process P1 was required for P2 to be able to complete.

Artifact Derived from Artifact: An edge [*was derived from*] from artifact A2 to artifact A1 is a causal relationship that indicates that artifact A1 should have been generated for A2 to be generated. 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 that indicates that agent Ag controlled the start and end of process P.

Role: Designates an artifact or agent's function in a process.

In Figure 1, the edge [*used*] say that a process used an artifact, while the [*was generated by*] edge 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, many artifacts may have been generated by a process, 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 just to distinguish the involvement of artifacts in processes.

The edge [*was controlled by*] expresses that a process was caused by an agent, essentially acting as a catalyst or controller. Since a process may have been controlled by several agents, their roles are also identified as controllers. This type of dependency represents a control relationship and not a data derivation. The edge [*derived from*] assert that artifact A2 was derived from another artifact A1, giving an oriented dataflow view of the provenance. In contrast to the edge [*was derived from*], an edge [*was triggered by*] allows a process to have an oriented view of past executions.

Moreover, the Open 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 this 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 allows for 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 indicate 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 marks when the process started and terminated. For [*was derived from*] and [*was triggered by*] edges, one timestamp 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 caused P2 to happen.

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.

For completion rules, there is the artifact elimination, also known as forward transformation. Figure 2 shows such transformation. The edge [*was triggered by*] can be obtained from the existence of [*used*] and [*was generated by*] edges. Also in the same figure, there is another completion rule, called artifact introduction, which establishes that the [*was triggered by*] edge is hiding the existence of an artifact used by P2 and generated by P1. The completion rules allow the establishment of the existence of some artifacts but it does not make explicit their identities. This is the consequence of using [*was triggered by*], which is a composition of [*used*] and [*was generated by*]. On the other hand, Figure 3 presents a completion rule regarding *process introduction*. The edge [*was derived from*] hides the presence of an intermediary process. However, the converse rule does not work without some internal knowledge of P, which is fundamental to ascertain if there is an actual dependency between A1 and A2.

When users want to find out the causes of an artifact or a process, their interest is in indirect causes that involve multiple transitions. For this purpose, a set of new relationships was created.

Multi-step "wasDerivedFrom": An artifact *a1* was derived from *A2*(possibly using multiple steps), written

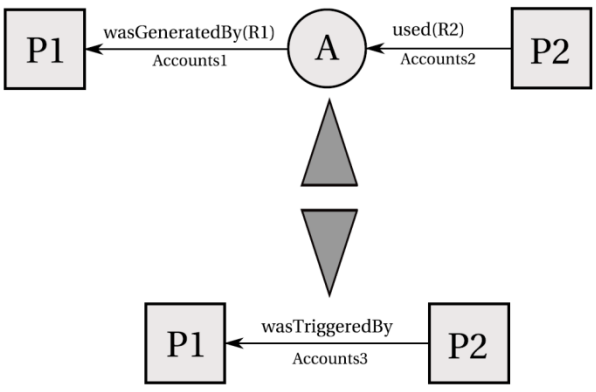

Figure 2: Artifact introduction and elimination. Source: [Moreau et al. 2011].

as *a1* →* *a2*, if *a1* was derived from *a2* or from an artifact that was itself derived from *a2* (possibly using multiple steps). In other words, it is the transitive closure of the edge [*was derived from*]. It expresses that artifact *a2* had an influence on artifact *a1*.

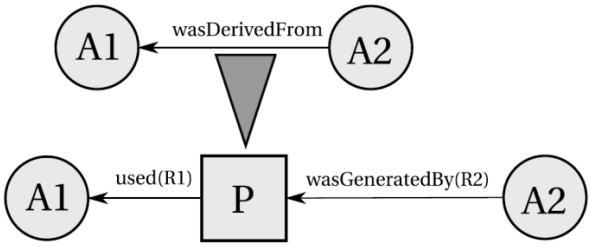

Figure 3: Process introduction. Source: [Moreau et al. 2011].

Secondary Multi-Step Edges:

Process *p* used artifact *a* (possibly using multiple steps): written *p* →* *a*, if *p* used an artifact *a* or an artifact that derived *a* (possibly using multiple steps).

Artifact a was generated by process p (possibly using multiple steps): written *a* →* *p*, if a or an artifact that derived *a* (possibly using multiple steps) that was generated by *p*.

Process p_1 was triggered by process p_2 (possibly using multiple steps): written p_1 →* p_2, if p_1 used an artifact that was generated or was derived from an artifact (possibly using multiple steps) that was itself generated by p_2.

Multi-step edges can be inferred from single step edges by eliminating artifacts that occur in chains of dependencies. Analyzing Figure 4, it is possible to infer that process *p2* was triggered by *p1*, omitting the fact that *p2* used *a3*, which was derived from *a2* that in turn was derived from *a1*, which was generated by *p1*. Other inferences are also illustrated in Figure 4.

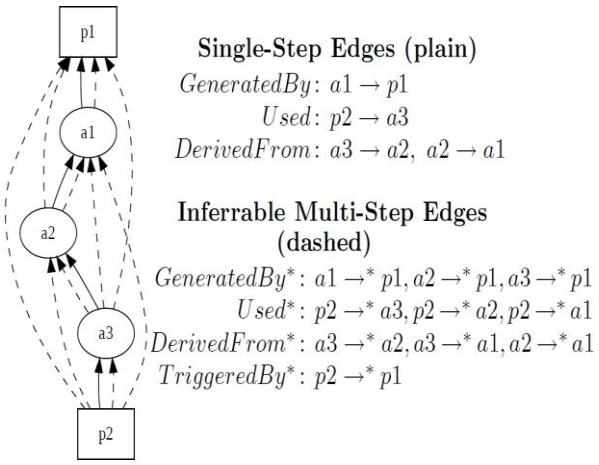

Figure 4: Inference. Source: [Moreau et al. 2011].

3 Provenance in Games

In this work we propose the adoption of provenance in the context of games. For this, it is necessary to map each node of a provenance graph to elements that can be represented in the game. As was mentioned earlier, the Open Provenance Model has three types of nodes: *Artifacts*, *Process* and *Agents*. In order to map them, it is necessary to find similarities in a game context.

Starting with *Artifacts*, their provenance definition states that they are "*an immutable piece of state that can represent a physical object* […]". Its definition already gives a clue on which role they can represent in the game context: objects. An object can be anything used in the game, for example in the case of an RPG, *Artifacts* can represent weapons, potions, legendary artifacts, magical objects, etc. It can represent anything meaningful to the development of the game history.

On the other hand, *agents* "*are contextual entities acting as a catalyst of a process that can enable, facilitate, control or affect its execution*". In a game context, agents can be mapped as people represented in the game, non-playable characters (NPCs), monsters, and players.

Lastly, *Processes* according to its definition are "*actions or a sequence of actions performed or caused by artifacts* […]". So, in a game context, *Processes* can be viewed as actions or events made by living or intelligent entities that are present in the game. Note that it was made a difference between living and intelligent. This difference is important to mention because, for example, in an RPG environment a sword can be expressed as an agent because this sword has an intelligence on its own. Despite being an object (sword), it can think and by an extent act, therefore it cannot be considered only as an object. It can also be as complex as being both an object and an agent at the same time.

Now, with all three types of nodes mapped into the game context, it is also necessary to map their causal relations to create the provenance graph. The Open Provenance Model defines a few causal relations which can be used similarly to their original context, but can be extended to be more suitable to the game context if necessary. Also, the Open Provenance Model can deal well with the aspect of time, which can be heavily explored in games, especially on games focused on storytelling, recording when each event happened and using this information to generate other events.

To generate actions and control events, each NPC in the game will require a decision tree in order to control his actions, providing an array of behavior possibilities. Event triggers can also be controlled by decisions tree. The next subsection describe which information are stored in actions, events, objects, and

agents. We also describe how the impact decisions tree can be achieved by actions, and how this information can be processed in order allow further provenance analysis.

3.1 Data model

Actions can be represented by a series of attributes that describe it and the context it was involved, allowing the creation of a provenance graph. As illustrated by Figure 5, 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, as discussed in this paper, 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 it affect are recorded for further analysis.

Events also work in a similar way as action, 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 be stored to aid in the construction of the graph. Lastly, agents can have their names, attributes, goals, and current location recorded. Figure 5 illustrates this model.

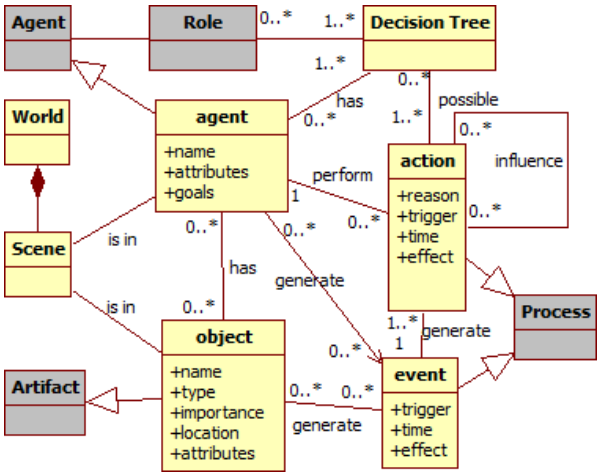

Figure 5: Data model diagram. Gray classes represents provenance classes.

3.2 Decision tree

For the purpose of controlling actions, each important NPC requires a decision tree that will be consulted for determining which action must be executed, similarly to a state diagram modeling. However, using decisions tree allows a greater variety of possible actions to be executed to reach the same goal, with different ways to reach it, which is only limited by its size and complexity of the tree.

Decision trees [Moret 1982] are a visual tool used to model decisions and their consequences, including probabilities of occurrence of events, resource costs, and usefulness of a particular outcome. This model can be considered as a deterministic algorithm to decide which variable to test based on the variables already tested and the results of its evaluation. Decision trees are represented by oriented tree format graphs composed of three distinct node types: decision; uncertainty; and terminal.

The usage of decisions trees brings a variety of actions and creates a diversity of possible outcomes in games, which can easily be traced to the reasons behind the outcomes by following the decision tree graph for each action. This information derived from decisions tree can be used for provenance in a novel manner.

3.3 Provenance Model

In order to store all the necessary data to be used later for provenance reasons, it is required a storage structure. Depending on the information structure, it is possible to use the structure itself for inference in provenance, simplifying some unnecessary information.

Considering the generation of actions, which are executed by an entity, the action information can be stored in a list. Each entity will then have a list of actions that contains all executed actions. This allows inferring who executed each action by simply looking at whose list it belongs to, without the need to explicitly say who executed the action. For event analysis it is possible to use an analogous approach. In the case there was an external influence that resulted in the triggering of an action, then the generated action is linked to the influence, which also has links to the actions that generated the influence. Since actions belong to lists that are linked to entities, then it is possible to infer who influenced the outcome of the action by following the links.

Entities present in a scene, or place, can be represented in a similar way as actions. Each scene has a list of entities that belong to it. To represent a world, a list of scenes is created, which in turn contains list of entities that are in the scene. Each entity in turn has a list of performed actions, which have links to influences. Using this structure, it is possible to simplify some inferences in the provenance model, such as to show only relevant actions, which has external influences, to evaluate the outcome of a game session. An example of such structure is shown at Figure 6, where the world has a list of scenes, each scene a list of all entities, and lastly each entity has a list of performed actions.

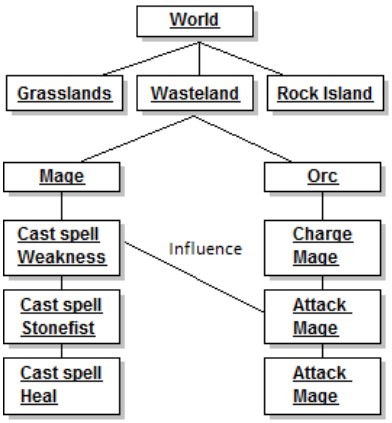

Figure 6: Example of structure

3.4 Provenance Analysis

The purpose of collecting information during a game session is to be able to use provenance techniques to analyze and infer the reasons of the outcome. In the previous sections, we introduced a framework to store such information. However, not all stored information is relevant for the analysis. The provenance graph contains replication of actions that did not provoke any significant change. These elements act as noise and can be omitted during provenance analysis by using completion and inference rules.

With the aim of finding actions that had an impact in the story, the actions that did not cause any dramatic change are omitted using multi-step inference rules. As an example, we may have a player in combat with an enemy and only after a few rounds it falls under the player's attacks. With the proposed framework, every round creates a node to represent the action taken by the player, which is attacking the enemy. This causes replication of data that is unnecessary for analysis, so it is possible to reduce all these individual attack nodes to simply one node.

However, that is not always true. The player could have made other actions against the enemy, which are also considered a form of attack, such as casting a spell, or a special attack maneuver, or even healing himself in order to survive. These actions are not duplicated, but can still be encapsulated for analysis. Since provenance is an analysis from the present to the past, the outcome of the battle is already known and can be used to decide which actions were relevant. 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 to just one node saying that the player attacked the enemy and was victorious. However, if the combat was challenging or the player lost, it is interesting to preserve the action nodes for analysis so the player can deeply understand the combat and decide what and when something went wrong.

4 Evaluation

The proposed framework was instantiated in a Software Engineering educational game named *Software Development Manager* (SDM) [Omitted for Blind Review]. The goal of SDM is to allow undergraduate students to understand the existing cause-effect relationships in software development. 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.

In SDM, which was developed using the game engine Unity3D [Higgins 2010], 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 have 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 (manager, analyst, designer, programmer, etc.), which valorizes attributes used to calculate the employee's 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 show 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.

4.1 Adapting SDM for the proposed framework

Some changes were made in the SDM game to introduce decision trees, allowing a variety of tasks and their respective actions, and a way to record all actions made by the player's employees for future usage on provenance. With these changes, it is possible to create an oriented graph representing the flow of actions

performed by each employee during the development of the software. The purpose of this graph is to use provenance techniques, presented earlier in this paper, allowing the player to view all the actions made during the playing session. With this information, the player can analyze the flow of the game and understand why the game session ended the way it did.

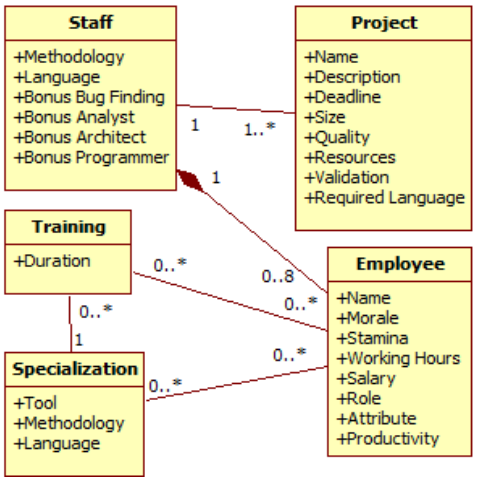

Figure 7: SDM's simplified class diagram. Adapted from [Omitted for Blind Review].

The Analyst has three different tasks to perform: Elicitation and validation; Requirements specification; and the creation of acceptance test cases. Another change was the way the analyst role works. Now, with the separated tasks of elicitation and specification, it is necessary to discover the system requirements by the process of elicitation and then create the model via specification. With these changes, the analyst role has four possible tasks, each one with its own actions: Elicitation and Validation, Specification, Quality, and a balanced task, which performs both elicitation and specification. These analysts tasks are illustrated in Figure 8.

For the Architect role, new tasks were introduced, which are responsible for creating integration and system test cases, generating prototypes to be used by the analyst, and his task of aiding programmers by working the software architecture.

The manager role was revised and changed as follows: He has the task of managing the staff and decides which role each employee should perform; he also decides the development focus, which can be Analysis, Development, Quality, and Balanced; finally, he decides the staff working hours and manage the hiring of new employees.

The roles of Programmer and Tester had suffered changed that affect each other and it is not the tester's responsibility anymore to fix bugs. The tester only finds and reports bugs so the programmer can fix them. Because of that, the programmer's tasks are as follow: Software Repair; Software Development; Code Refactoring. Moreover, the tester only task is to report bugs found by the execution of test cases.

Figure 8: Analyst Decision Tree Example. Orange boxes represent end nodes (tasks). Red boxes are value evaluation. Green lines represent probabilistic paths and blue lines are decision paths.

With the new programmer's task of refactoring, a new aspect was introduced in the software development, which is the quality of the code. This quality influences the probability of removing and introducing bugs in the software. Also, the quality of the code is directly affect by how the programmer is working, that now has three different ways: Ad hoc; Design-Code; and Test-Driven. Only the first one affects quality, done in a negative way. To increase the code quality it is necessary to do refactoring of parts in the software already implemented.

The Draw-Code mode of programming is the default one and equivalent to the one in the previous version of the game. Test-driven allows the programmer to develop the software with minimal chances of introducing new bugs because the programmer is taking his time to create unitary test cases, check the code for bugs and repair.

With these changes in roles, other changes were made in the structure of the game to accommodate them. The first change was related to test cases and software bugs. Because of different test cases available and performed by different roles, it was necessary to expand the way bugs are represented in the game. As such, there are now four categories of bugs: acceptance, system, integration, and unitary.

Figure 9 illustrates the changes made in each role and allows the player to configure the tasks of each employee. The decisions trees for each role use all options presented in that screen. Due to the overwhelming decisions allowed for the player to configure his staff, setting roles and tasks for each employee, the staff manager can unburden the player by deciding the staff configuration in case the player does not want to micromanage the game, giving some of the responsibility to the staff manager. Doing so, the manager will distribute roles and tasks for each employee depending on the development progress, which can be determined by the manager or the player, depending on the degree of autonomy given to the manager.

Another change made in the game is to allow an employee to perform up to two roles simultaneously, having a primary and secondary role. This change was based on the fact that in provenance, when performed an action, the role of the agent can be relevant, distinguishing involvement of artifacts and agents in processes. When an employee has both roles filled, the player or the staff manager decides the rates for each role. In other words, it specifies how many hours of his time that employee dedicates for each role. All rules for the primary role apply to the secondary role, and the productivity of the primary and secondary roles are multiplied by their rate factor. The staff manager also can use this feature for assigning roles.

Figure 9: Task Configuration window

With the revised roles and their respective tasks, decision trees were made to allow for a task selection and create diversity on the game flow. Each decision tree corresponds to a role, obeying their respective tasks. However, the way these tasks are performed may vary depending on the situation. As such, the decision process is influenced by internal reasons, generated by the employee himself, and external reasons, decisions made by the player or staff manager. Figure 8 illustrates an example of such decision tree, belonging to the analyst role and Figure 9 illustrates the external reasons.

The introduction of decision trees allows the variety of actions performed by each role. These actions, which are the result of a path from root to leaf in the decision tree, are stored for future provenance analysis, along with the path taken as well as the actions that influenced it. When an action that generates influence is executed, is stored the type of action it influences and a pointer to it. This stored information is used every time a new action is executed while the influence persists, identifying external influences. Other tasks can also produce actions for storage, such as hiring and firing an employee, training, player choices and decisions.

4.2 Information Structure

The information structure used on SDM is similar to the one explained in section 3.3. As such, each project is a scene contains a list of all entities that participated in it. These entities are employees that worked in the project and the player. Each employee has a list of actions made and each action contains its details, including links to other actions in case of external influences. Figure 10 illustrates the action nodes

generated during the game. These actions have details about who performed it, when it was performed, which task generated it, if there were any external influences, and a description of the decision tree path taken to generate the action.

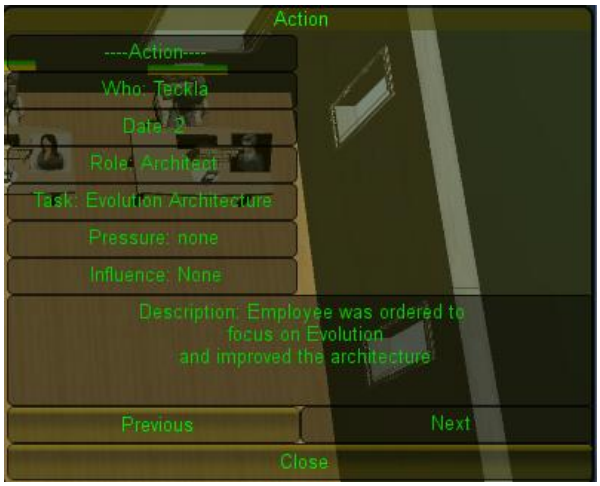

Figure 10: Action details

As said, all actions are grouped in the owner list, meaning that each employee has a list of actions. The player also has a list of all actions performed. Figure 11 illustrates the information organization for a project, showing all the employees involved in it and the details of the project.

Figure 11: Information Organization

Each employee slot in the picture is a list of all employees that belonged to that slot and in the right hand side of each slot is the action list, showing the last action performed. By selecting the action, it shows its details, as depicted in Figure 10, and transverses the list by the Previous and Next buttons. It is analogous for the employee list.

4.3 Provenance Analysis in SDM

With the adaptations in SDM, it is now possible to use the collected data for provenance analysis. However, due to limitation on Unity3D, the data should be exported for an external visualization and analysis tool, which will remove unnecessary information, duplicate actions or similar ones by inference rules. For the purpose of the game, the only interesting actions are the ones that influence or are influenced by other actions, such as player and manager decisions or tasks that generate interference on other roles like architecture task from an architect.

Action that does not generate influence or does not influence other actions are not relevant for the analysis, due to the fact the action did not change the state of development, negatively or positively. Nevertheless, it is important not to forget that even if they are not relevant for the analysis, they may have been relevant for the development of the software in the game. Without such actions, the game would stagnate and would not progress. The problem is not these actions, but the decisions made for the execution of these actions.

After cleaning the data, the information is more adequate for analysis and provenance inferences. This way, the player is able to trace actions that had an impact during development and study the adequacy of his decisions and the course of actions that lead from these decisions. Identifying these actions is essential for understanding why something happened the way it did. This refined action graph can be displayed for the player by external tools designed for graph display, aiding visually the analysis.

Understanding the reasons of the outcome, the player is able to learn from his decisions and analyze more efficient ways to develop future projects. In addition, it allows the perception of mistakes made that should be avoided in the future.

5 Conclusion

This paper proposed a new framework for provenance in games, allowing post game analysis to discover divergence points that contributed to the end result of the gaming session. This framework can be used on serious games to improve understanding by analyzing game flow and identifying actions that influenced the outcome, aiding the player to understand why it happened the way it did. While many concepts of provenance were directly used, for this work we also proposed new elements and adaptions for a video-game software.

This paper also showed a game in which our proposed framework was instantiated, collecting the necessary information for post analysis using

provenance. However, due to the complexity of data extraction, the usage of provenance was not executed but is planned as future work, exporting all collected data, generate a graph and apply provenance techniques for the game analysis.

Acknowledgements

Section omitted for Blind Review.

References

Andersen, E. et al., 2010. Gameplay analysis through state projection. *In: ACM Press*, pp.1–8.

Baker, A., Navarro, E. and van der Hoek, A., 2003. Problems and Programmers: An Educational Software Engineering Card Game. *In: International Conference on Software Engineering*, pp.614–621.

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

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

Consalvo, M. and Dutton, N., 2006. Game analysis: Developing a methodological toolkit for the qualitative study of games. *In: Game Studies*, v. 6.

Dantas, A., Barros, M. and Werner, C., 2004. Treinamento Experimental com Jogos de Simulação para Gerentes de Projeto de Software. *In: Simpósio Brasileiro de Engenharia Software*, v. 18, pp.23–38.

Figueiredo, K. et al., 2010. Jogo de Estratégia de Gerência de Configuração. *In: III Fórum de Educação em Engenharia de Software*.

Higgins, T., 2010. Unity - 3D Game Engine. Available at: http://unity3d.com/ [Accessed May 5, 2011].

OMMITED FOR BLIND REVIEW

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

Moret, B., 1982. Decision Trees and Diagrams. *In: ACM Computing Surveys (CSUR)*, v. 14(4), pp.593–623.

Navarro, E. and van der Hoek, A., 2004. SIMSE: An Interactive Simulation Game for Software Engineering Education. *In: Proceeding of CATE*, p.233–233.

Warren, C., 2011. Game Analysis Using Resource-Infrastructure-Action Flow. *Fícial*. Available at: http://fícial.wordpress.com/2011/10/23/game-analysis-using-resource-infrastructure-action-flow/ [Accessed July 3, 2012].