*Visualizing Software Engineering Learning Sessions Through 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*—**Software engineering is an area of computer science that focuses on practical and theoretical aspects of the software production. Teaching software engineering is usually done by theoretic classes with few practical exercises. To circumvent this problem, a raising approach consists on the adoption of games. In software engineering games, decisions are key factors to transmit knowledge. However, mistakes made by wrong decisions may jeopardize the learning process, especially when reproducing its effects is not a viable option. With this in mind, we introduce a novel approach based on provenance concepts to present the decisions and effects of such decisions when learning via games. We extract provenance during software engineering game execution and generate a provenance graph that can be used for a broader range of analysis.**

***Keywords-component; Software Engineering; Provenance; Education.***

# Introduction

In Software Engineering, the traditional teaching consists of lectures and practical work with the intent of using theory learned in the class in order to aid understanding. However, these practical works usually do not stimulate the student’s interest. In order to solve this problem, games [1] have been used for aiding students to learn and understand concepts taught in classrooms [3, 18] by stimulating curiosity and providing motivation for learning [22].

However, 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 game play may be useful for understanding the achieved results. This analysis is also fundamental for detecting symptoms of problems that occurred due to wrong decision-making and to know if the student learned the concepts present in the game.

Without the game flow analysis, the student would be required to play the game again and make different decisions to intuitively guess which ones were not relevant to the situation. However, depending on the game dynamics and its complexity, reproducing the same state can be unviable, making it difficult to try new solutions.

With this in mind, the goal of this paper is to improve the software engineering learning process by making the tacit knowledge explicit as insights on how the game session progressed. This is achieved by analyzing the game flow data using provenance[[1]](#footnote-1). The provenance analysis process collects data and generates a provenance graph, relating actions, decisions, and events that occurred throughout the game in a high level model. This allows a broader range of analysis from both the student and the teacher. For instance, the provenance graph allows the student to browse the data, identifying actions that influenced in the outcome. It also helps to understand how events were generated and which decisions contributed to them. This process also aids in the identification of mistakes, allowing the user to reflect upon them for future interactions.

In our previous work [14], we introduced the usage of digital provenance [10] 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. 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 and analysis methodology, allowing the support of a broader range of analysis over a software engineering game named SDM [15].

SDM focuses on introducing Software Engineering concepts and skills to undergraduate students. The provenance gathering and analysis model presented in this paper allows students and tutors to visualize the game flow and identify steps that lead to successful or unsuccessful outcome. We also believe that the ideas discussed in this paper can open a wide range of research in the field of behavior patterns data mining of the learning sessions.

This paper is organized as follows: Section II provides related work in the area of software engineering games and game flow analysis. Section III provides a background on provenance and Section IV introduces our framework for provenance gathering. Section V presents our approach for provenance visualization through graphs. Section VI presents the adoption of provenance visualization in software engineering. Finally, Section presents the conclusions of this work and points out some future work.

# Related Work

Our work provides support to understand how students performed when learning via software engineering games. This way, our related works is dual: software engineering games and game flow analysis. Our next three paragraphs introduce two popular software engineering games and the rest of the section discusses the existing support for game flow analyses.

In [3], the authors present a software engineering card game called *Problems and Programmers*. The focus is teaching software engineering through a simulation of the software development process from conception to completion. The players learn tactics to avoid problems during the development of the product while, at the same time, competing with each other in order to complete their respective products in less time. It rewards those that follow software engineering concepts and penalize those that try quicker and riskier approaches.

In [18], the authors present a simulation game called *SimSE*. In it, the player assumes the position of a project manager and has to manage the software development. The fundamental goal of this game is allowing customization of the simulated process model to be used by tutors during the presentation of content related to software life cycle.

These two games, among many others, assume a “learn by experience” process, but none support on-the-fly or post-mortem game flow analysis. This type of game flow analysis is a key element to validate assumptions created by the student during the “learn by experience” process.

In the digital game domain, Warren [11] 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 [7] 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 [11], 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 [5] was presented in [19]. This method organize the story using PNF networks [20], 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) [16] was created at the *Provenance Challenge*, which is a collocated event of IPAW. Recently, another provenance model was developed, named PROV [23], 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 [16], 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 represent a causal dependency between the source, which denotes the effect, and the destination, which denotes the cause.

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 [14] 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*" [16], 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*" [16], 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*" [16], were mapped to actions or events made by entities in the game.



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

In [14], the generation of actions and events are controlled by decision trees [17]. 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. Moreover, in [14] 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 . Besides this, [14] also presented an information structure to store collected game data to generate a game flow log for provenance analysis.

# Provenance Visualization

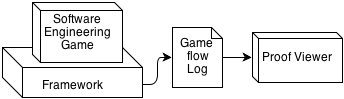


Figure 2. Relationships between a software engineering game using *provenance in games* framework and the *Proof Viewer*.

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 [13] 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*. 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 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 3. Example of a generated provenance graph.

A small example of a generated provenance graph from exported data is illustrated by . Following the provenance notation specification, each node shape in is related to its type. Square nodes represent *process* nodes, circles are *artifacts* nodes and an octagon represents *agent* nodes. 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 . 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 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. To emphasize the neutral relationships lack of importance, they are also dotted. These edge types are illustrated in .

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. 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.

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 [8], 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 nodes that contain a player financial value will have their colors changed according to its value. Activating this type of filter allow the user 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 V 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 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 that the development process has its information recorded in a daily basis. With the framework proposed in [14], every day generates nodes for each action executed during that day. This may generate data that is unnecessary for analysis, so it is possible to collapse all individual nodes from the same week to a unique node, summarizing that week’s work. Another case could consist in sequences of similar actions, like an employee testing the software during an entire week. In this case, *Proof Viewer* could omit all nodes and represent his week progress with only one node. Note that all collected information is preserved and the only change is on how data is displayed.

Since provenance is analyzed from the present to the past, the outcome is already known and can be used to decide if the actions of a specific week are relevant or not. If during the week nothing out of ordinary happened, then it can be simplified into just one node representing the general progress. However, if the week’s progress is very different from the others, it can be interesting to show all action nodes for analysis, allowing the player to identify important facts that influenced the overall outcome.

Note, however, that *Proof Viewer* does not provide inference for the user, but only the necessary means 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 [4, 6, 9, 12, 24].

# Provenance Visualization in Software Engineering

We instantiated this provenance analysis infrastructure, which uses the proposed framework presented in [14], in a Software Engineering educational strategy game named SDM (*Software Development Manager*) [15]. 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.

In SDM 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.

## Information Storage

The information structure used on SDM is similar to the one explained in [14]. 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 [14] and illustrated by . 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 represent Prototypes, Test Cases, and Project. 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 .

## Provenance Graph

With the adaptations made in the original SDM concepts [14], 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 4. An example of credits status filter.

By analyzing the graph, it is possible to reach some conclusions of why the story progressed the way it did. As an example[[2]](#footnote-2), illustrates a scenario where the player had financial problems. To simplify the picture, some collapses were made, omitting most of the *agent’s* *processes*. The *artifacts* represent instances of the development stage, and are colored according to the player’s financial condition. The *processes* present in the picture represent hiring actions in gray and resignations in brown.

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 , 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.



Figure 5. Non-collapsed graph from using filter: Morale.

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. 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.



Figure 6. Example of a provenance graph analysis.

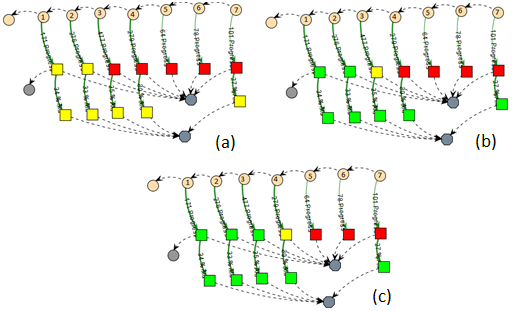


Figure 7. Graph from Figure 6 using filters: working hours (a), stamina (b), and morale (c).

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 .

In 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.

By changing the filter again to show stamina levels, we can see in that in node 3 his stamina 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. 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 introduces new perspectives on software engineering learning, leveraging the current state of the art, based on game, 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 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.

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 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 running experimental studies on the usage of provenance in educational games to evaluate the aspects of learnability.

##### Acknowledgment

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., Liu, Y.-E., Apter, E., Boucher-Genesse, F. and Popović, Z. 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] Baker, A., Navarro, E. and Van der Hoek, A. 2003. Problems and Programmers: An Educational Software Engineering Card Game. *International Conference on Software Engineering(ICSE)*. (2003), 614–621.

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

[5] Cavazza, M., Charles, F. and Mead, S.J. 2002. Character-based interactive storytelling. *IEEE Intelligent Systems*. 17, 4 (Aug. 2002), 17 – 24.

[6] Cios, K., Pedrycz, W. and Swiniarski, R.W. 1998. *Data mining methods for knowledge discovery*. Kluwer Academic Publishers.

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

[8] Diehl, S. 2007. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer.

[9] Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. 1996. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*. 17, 3 (Mar. 1996), 37.

[10] Freire, J., Koop, D., Santos, E. and Silva, C.T. 2008. Provenance for Computational Tasks: A Survey. *Computing in Science Engineering*. 10, 3 (Jun. 2008), 11 –21.

[11] 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.

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

[13] Joshua O’Madadhain, Danyel Fisher and Tom Nelson 2010. *JUNG: Java Universal Network/Graph Framework*. Open-source.

[14] Kohwalter, T., Clua, E. and Murta, L. 2012. Provenance in Games. *2012 XI Brazilian Symposium on Games and Digital Entertainment (SBGAMES)* (Brasilia, Nov. 2012).

[15] Kohwalter, T., Clua, E. and Murta, L. 2011. SDM – An Educational Game for Software Engineering. *2011 X Brazilian Symposium on Games and Digital Entertainment (SBGAMES)* (Salvador, Nov. 2011).

[16] 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. and Den Bussche, J.V. 2011. The Open Provenance Model core specification (v1.1). *In: Future Generation Computer Systems*. 27, 6 (Jun. 2011), 743–756.

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

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

[19] Passos, E.B., Montenegro, A.A., Clua, E.G., Pozzer, C.T. and Da Silva, F.S.C. 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.

[20] 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.

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

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

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

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

1. 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 [21]. [↑](#footnote-ref-1)
2. 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-2)