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.** 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.

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

1. Introduction

The conclusion of a game session derives from a series of decisions and actions made throughout the game. In many situations, analyzing and understanding the events, mistakes, and flows of a concrete gameplay experience may be useful for understanding the achieved results. A game flow analysis might be fundamental for detecting symptoms of problems that occurred due to wrong decision-making or even bad gameplay design. Without this type of analysis, the player would be required to play the game again and make different decisions to intuitively guess which ones were not adequate to the situation. However, depending on the game dynamics and its complexity, reproducing the same state can be unviable, making it difficult to replay and try new solutions.

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 [2, 3] due to their stimulating curiosity characteristic and for providing motivation for learning [4]. Understanding the educational results obtained in a serious game is important to assimilate the knowledge and concepts passed in the game. In addition, examining the game flow allows the identification of good and bad attitudes made by the player. This knowledge can be used in future game sessions to avoid making the same mistakes or even to adjust gameplay features.

The goal of this paper is to improve the player’s understanding of the game flow, providing insights on how the story progressed and influences the outcome. In order to improve understanding, we provide the means to analyze the game flow by using provenance. The provenance analysis is done by processing the collected gameplay data and generating a provenance graph, which relate the actions and events that occurred during the game session. This provenance graph allows the 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 our previous work [5], we introduced the usage of digital provenance [6] 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.

In this work, the *Provenance in Games* framework is instantiated in the SDM game [7] 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 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.

1. 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 [2], 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 [3], 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 [8] 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 [9] 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 [10], offers a way to visually analyze play steps, providing detailed visual representation of the actions taken by the player through the game.

Besides [8], 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 [11] was presented in [12]. This method organize the story using PNF networks [13], representing the temporal structure of the events that make up the plot. This structure can also be used in the generation of new events to the story, but is restricted to temporal coherence between the game events, without providing insights of positive or negative reinforcements.

1. Provenance

Provenance is well understood in the context of art or digital libraries, where it respectively refers to the documented history of an art object, or the documentation of processes in a digital object's life cycle. In 2006, at the International Provenance and Annotation Workshop (IPAW), the participants were interested in the issues of data provenance, documentation, derivation, and annotation. As a result, the Open Provenance Model (OPM) [14] was created at the Provenance Challenge, which is a collocated event of IPAW. Recently, another provenance model was developed, named PROV [15], 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 [14], 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 immutable entities 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.

1. 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 [5] 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" [14], 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*" [14], 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*" [14], were mapped to actions or events made by entities in the game.

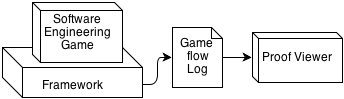
In [5], the generation of actions and events are controlled by decision trees [16]. However, any decision making algorithm can be used instead to control actions and behaviors. These generated actions and events are represented as *processes* nodes in the provenance graph. Moreover, in [5] we proposed 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, [5] also presented an information structure to store collected game data to generate a game flow log for provenance analysis.



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

1. Provenance Visualization

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 [17] 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.



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

First, the game flow log, which contains game events, is processed and used to generate a provenance graph for analysis. After that, our tool creates the graph’s edges and 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.



**Fig. .** 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 data, not all collected information is relevant for every type of analysis. Thus, the provenance graph might contain actions that did not provoke any significant change. These elements act as noise and can be omitted during analysis. To do this, it is possible to collapse 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*, improving visibility of 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 [18], 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 VI 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 on a daily basis. With the framework proposed in [5], 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 involves other areas of research [19–23].

1. Provenance Visualization in Software Engineering

We instantiated this provenance analysis infrastructure, which uses the proposed framework presented in [5], in a Software Engineering educational strategy game named SDM (Software Development Manager) [7]. The goal of SDM is to allow undergraduate students to understand the existing cause-effect relationships in the software development process. Thus, the adoption of provenance becomes an important instrument to better support knowledge acquisition, allowing the possibility of tracking mistakes made during a game session.

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 in accordance with the product quality.

* 1. Information Storage

The information structure used on SDM is similar to the one explained in [5]. 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 [5] and illustrated by . Each node contains different information according to its type.

Processes nodes, which represent actions executed during the game by employees, store information about its execution. This information includes who executed it, which task and role the employee was occupying, as well as the current morale and stamina status. Worked hours and credits spent to execute the action are also stored. Lastly, the progress made in his task and a description of the action, explaining his decision making process. Besides those, if the action had any external influences, or used or altered an artifact, a link to the action that affects its execution and the artifact is included.

Agent 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 .

* 1. Provenance Graph

With the adaptations made in the original SDM concepts [5], 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.

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), 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.



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

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 loses 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. illustrates an example scenario. To simplify the graph visualization due to size limits, we focus only on two *agents* and the main *artifact* known as “project”. Those *agents’* roles are programmer and manager, with the manager acting as a supporting role for the programmer.



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

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.

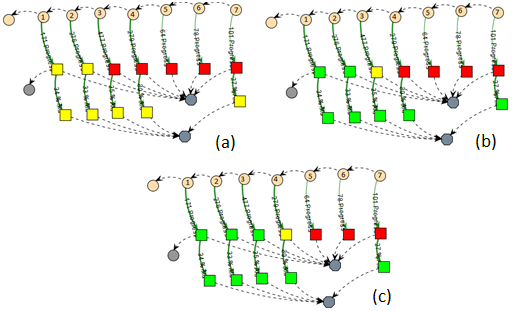


**Fig. .** Example of a provenance graph analysis.

In node 2, the programmer 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.



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

1. Conclusion

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

The provenance visualization can occur both on-the-fly or in post-mortem sessions. It allows the discovery of issues that contributed to specific game flows and results achieved throughout the gaming session. This analysis can be used on games to improve understanding of the game flow and identifying actions that influenced the outcome, aiding 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.

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

1. References

1. Abt, C.C.: Serious Games. University Press of America, Abt Books (1987).

2. Baker, A., Navarro, E., van der Hoek, A.: Problems and Programmers: An Educational Software Engineering Card Game. International Conference on Software Engineering(ICSE). 614–621 (2003).

3. Navarro, E.: Simse: a software engineering simulation environment for software process education, (2006).

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

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

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

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

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

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

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

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

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

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

14. Moreau, L., Clifford, B., Freire, J., Futrelle, J., Gil, Y., Groth, P., Kwasnikowska, N., Miles, S., Missier, P., Myers, J., Plale, B., Simmhan, Y., Stephan, E., den Bussche, J.V.: The Open Provenance Model core specification (v1.1). In: Future Generation Computer Systems. 27, 743–756 (2007).

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

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

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

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

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

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

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

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

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

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