PinGA: Semi-Automated Provenance Instrumentation for Digital Games

David Lopes de Santana Vazquez[1], **Troy Costa Kohwalter**[1]

[1] Instituto de Computação – Universidade Federal Fluminense (UFF)
Niterói – RJ – Brazil

davidvazquez@id.uff.br, troy@ic.uff.br

Abstract. Introduction: *Game telemetry techniques commonly record events and state changes but fail to explicitly represent causal relationships between gameplay actions. Provenance-based approaches address this limitation by modeling cause-and-effect through provenance graphs; however, existing solutions such as the Provenance in Games (PinG) framework and its Unity implementation (PinGU) rely on extensive manual instrumentation, which limits scalability and adoption.* ***Objective:*** *This work proposes PinGA, an automation-oriented approach for provenance instrumentation in digital games. PinGA aims to reduce the manual effort required to configure provenance tracking while preserving the causal expressiveness of provenance-based gameplay analysis.* ***Methodology:*** *PinGA defines a semi-automated workflow based on three main steps: the discovery of monitorable gameplay elements, developer-guided semantic configuration of those elements in provenance terms, and runtime monitoring for provenance graph generation. To evaluate the feasibility of this proposal, PinGA was instantiated in the Godot game engine through PinGODOT and its editor-integrated automation layer, PinGODOT Manager.* ***Results:*** *A preliminary evaluation conducted on a 2D game prototype showed that the workflow operated end-to-end, shifted a substantial part of manual instrumentation to editor-based configuration, and generated provenance graphs that supported causal inspection of the evaluated gameplay session.*
Keywords *Game Analytics, Provenance, Godot, Automated Telemetry*

1. Introduction

Game telemetry plays a fundamental role in understanding player behavior, identifying design issues, and supporting balancing decisions. Traditional telemetry approaches typically focus on logging isolated events or state changes, such as player deaths, item collection, or score variation. Although useful, these approaches provide limited support for reasoning about causality, making it difficult to determine why certain outcomes occur during gameplay.

Provenance-based approaches address this limitation by explicitly representing cause-and-effect relationships between actions, entities, and agents. In digital games, the Provenance in Games (PinG) framework adapted provenance concepts to gameplay analysis through provenance graphs, allowing game sessions to be interpreted not only in terms of what happened, but also in terms of how events and states contributed to later outcomes [Kohwalter et al. 2012]. This perspective was later operationalized in PinGU, a Unity-based implementation that demonstrated the analytical usefulness of

provenance graphs for gameplay inspection and post-hoc analysis [Kohwalter et al. 2018, Kohwalter et al. 2017]. However, PinGU also exposed an important practical limitation: provenance collection depends heavily on manual instrumentation, requiring developers to insert tracking logic directly into gameplay code.

This manual effort introduces development overhead, increases coupling between telemetry and game logic, and makes provenance-based analytics harder to adopt in iterative development contexts such as rapid prototyping, indie production, and educational projects. In parallel, the Godot game engine has gained prominence as an open-source alternative in digital game development, with increasing relevance among independent developers and adoption in academic settings [Holfeld 2024, Gestwicki 2021]. Despite this growth, Godot still lacks native support for provenance-based telemetry.

This work addresses that gap through **PinGA**, a semi-automated approach for provenance instrumentation in digital games. Rather than centering the contribution on a simple adaptation of PinG to another game engine, PinGA focuses on making provenance collection more practical through a workflow based on the discovery, developer-guided semantic configuration, and runtime monitoring of gameplay elements.

To evaluate the feasibility of this proposal, we instantiate PinGA in the Godot game engine through **PinGODOT**, which provides the runtime provenance infrastructure required to represent, manage, and export provenance data in Godot. On top of this infrastructure, **PinGODOT Manager** provides the editor-integrated automation layer that operationalizes the PinGA workflow inside the engine. We also developed an in-engine provenance visualization solution for inspecting collected graphs directly inside the editor. A preliminary evaluation in a 2D prototype indicates that the proposed workflow preserves the analytical usefulness of provenance graphs while reducing instrumentation effort and integration overhead.

2. Background

Digital provenance concerns the representation of how data and results are produced, including the entities, activities, and agents involved in a process [Buneman et al. 2001, Moreau 2010]. These foundations were later consolidated in the W3C PROV Data Model (PROV-DM) [Moreau e Missier 2013], which provides a standardized way to describe causal and derivation relationships.

In digital games, the Provenance in Games (PinG) framework adapted these concepts to gameplay analysis by modeling game objects as Entities, player actions as Activities, and players or characters as Agents [Kohwalter et al. 2012]. This approach enables gameplay sessions to be represented as provenance graphs, making explicit not only what happened but also how gameplay events and states shaped later outcomes. Provenance-based analysis therefore offers a richer perspective than traditional telemetry, which often records events in isolation and provides limited support for causal interpretation.

This perspective was later implemented in practice through PinGU, a provenance-based solution for the Unity engine [Kohwalter et al. 2018, Kohwalter et al. 2017]. PinGU demonstrated the analytical usefulness of provenance graphs for post-hoc gameplay analysis, but also revealed an important limitation: provenance collection depends heavily on manual instrumentation, requiring developers to explicitly insert tracking logic into gameplay code.

3. Related Work

Game analytics has been extensively explored through event-based telemetry systems that log player actions, state changes, and performance metrics. Commercial game engines such as Unity and Unreal provide built-in analytics services, while academic research often relies on custom logging frameworks to collect gameplay data. Although effective for descriptive analysis and statistical reporting, these approaches typically treat events in isolation, offering limited support for reasoning about causal relationships between player actions and gameplay outcomes. Prior work in game analytics has focused on large-scale data collection and statistical techniques to define player models and personas through clustering and behavioral profiling [Drachen et al. 2009, Drachen et al. 2013].

To overcome the limitations of analysis based solely on event logs, provenance-based approaches have been proposed as a means of explicitly modeling cause-and-effect relationships in interactive systems. In the context of games, the Provenance in Games (PinG) framework introduced provenance graphs to represent gameplay sessions, enabling designers to trace outcomes such as player failure back to contributing actions and states [Kohwalter et al. 2012]. This approach was later operationalized in the Unity engine through PinGU, demonstrating that provenance-based telemetry supports richer post-hoc analysis and design reasoning than traditional event logs [Kohwalter et al. 2018, Kohwalter et al. 2017]. More recently, this line of research was extended through ProvReplay [Thurler et al. 2025], which combines provenance data with synchronized gameplay replay to support qualitative inspection of game sessions and provide richer gameplay context during analysis. Although this extension improves how collected provenance data can be explored and interpreted, it still depends on provenance capture pipelines derived from previous PinG-based solutions and therefore does not address the instrumentation burden through editor-integrated automation.

Recent work has explored more structured approaches to gameplay data collection and analysis, especially in contexts where telemetry must support reproducible analytics workflows across games and studies. In educational and research-oriented settings, Open Game Data proposes a modular pipeline and data standards for collecting, transforming, and reusing gameplay telemetry across multiple games and institutions, emphasizing interoperability and scalable analysis [Gagnon et al. 2024, Swanson e Gagnon 2024]. In a similar direction, analytics-by-design approaches have shown how telemetry collection can be embedded into the design of serious games to capture gameplay features systematically and support predictive analysis of player performance [Lu et al. 2023]. Other studies have highlighted the value of richer telemetry representations and playback-oriented analytics interfaces for post-mortem gameplay inspection [Mane e Elmqvist 2024], including web-based frameworks for 3D telemetry analysis in multiplayer simulation contexts. Although these approaches advance how gameplay data is collected, structured, and analyzed, they are generally not aimed at provenance-based causal modeling, nor do they provide an editor-integrated automation workflow for provenance instrumentation in modern game engines.

Despite these advances, the reviewed approaches do not provide a general automation-oriented model for provenance instrumentation that combines provenance-based analytics with reduced integration effort in modern game development workflows. This gap motivates the proposal of **PinGA**, a semi-automated approach that preserves the ana-

lytical benefits of provenance graphs while shifting part of the instrumentation burden from source-code modification to discovery, configuration, and runtime monitoring mechanisms.

In this work, PinGA is instantiated in the Godot game engine, generating **PinGODOT** and its editor-integrated automation layer, **PinGODOT Manager**. Godot was selected due to its open-source nature and growing adoption among independent developers and academic contexts, where lightweight and accessible analytics tools are especially relevant.

4. PinGA: Automation Logic for Provenance in Games

PinGA (Provenance in Games Automation) can be understood as an automation-oriented perspective within the PinG framework, aimed at reducing the manual burden of provenance instrumentation in game development. Rather than redefining the provenance model established by PinG, it emphasizes the need to make provenance capture more practical and better integrated with development workflows.

At its core, PinGA formalizes a three-step automation logic for provenance instrumentation. First, **discovery**, in which potentially relevant gameplay elements – such as game objects, properties, and behavioral methods – are exposed for monitoring through introspection mechanisms, reducing the need for developers to manually enumerate them in the source code. Second, **semantic configuration**, in which the developer selects which of these discovered elements should be tracked and how they should be interpreted in provenance terms, such as Agent, Entity, Activity, or attribute. Third, **runtime monitoring**, in which the configured elements are monitored during gameplay execution and translated into provenance updates, allowing causal graphs to be generated with substantially less explicit per-action tracking code.

This model shifts a substantial part of provenance instrumentation from scattered source-code modifications to a more declarative, configuration-driven workflow. The developer still plays an essential role in assigning semantic meaning to the collected elements, but the operational effort required to connect provenance capture to gameplay logic is significantly reduced. In this sense, PinGA can be understood as a semi-automated provenance instrumentation model that preserves the causal expressiveness of PinG while improving its practical integration into development workflows.

An important aspect of this perspective is that automation does not imply full semantic autonomy. Although the system may identify technical opportunities for monitoring, it cannot always determine by itself which of them are analytically meaningful for gameplay analysis. For this reason, PinGA explicitly adopts a hybrid approach that combines automated discovery and runtime capture with developer-guided semantic configuration. This balance is essential to making provenance collection more practical without sacrificing interpretability.

Although PinGA reduces the amount of code-level instrumentation required to collect provenance data, the approach should not be interpreted as a fully automatic solution. Its automation focuses on three aspects: discovering monitorable elements within the game structure, exposing them for editor-based configuration, and monitoring selected elements at runtime to generate provenance updates. However, semantic decisions remain

developer-guided. The developer is still responsible for deciding which discovered elements are analytically relevant and defining which events or state changes should be interpreted as meaningful gameplay activities. PinGA defines the general semi-automated instrumentation logic, independent of a specific game engine. PinGODOT is the Godot-based provenance infrastructure, and PinGODOT Manager is the editor-integrated layer that operationalizes the PinGA workflow in Godot.

By separating this automation logic from any specific engine implementation, PinGA establishes an intermediate conceptual layer between provenance theory and engine-level tooling. In this paper, that logic is instantiated in the Godot ecosystem through PinGODOT and its editor-integrated automation extension, PinGODOT Manager. Figure 1 illustrates the architecture of this instantiation, highlighting the relationship between the PinGODOT core components and the PinGODOT Manager automation layer.

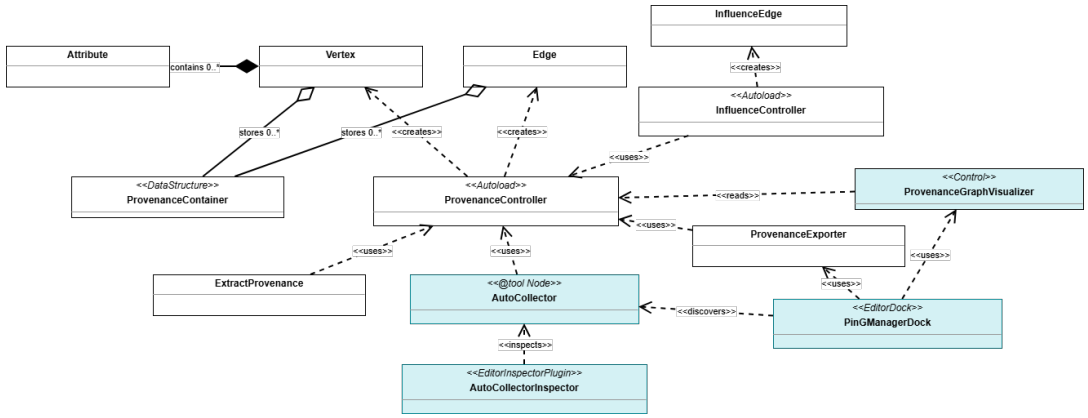

Figure 1. Architecture of the PinGA instantiation in Godot.

5. PinGODOT

This section presents the Godot-based realization of the automation logic introduced in Section 4. As illustrated in Figure 1, PinGODOT provides the engine-native provenance foundation, and PinGODOT Manager operationalizes this workflow inside Godot through editor-based configuration and runtime monitoring.

PinGODOT adapts the PinG conceptual model to Godot's node-based architecture through a layered design that separates provenance representation, runtime management, and automation support. The framework is organized into three layers: a **Core Layer** of AutoLoad singletons managing provenance graph consistency; an **Interface Layer** of reusable node components that allow gameplay objects to interact with the provenance system; and an **Automation Layer** introducing editor-level support for configuring and monitoring provenance collection.

PinGODOT implements Godot-native classes for provenance structures such as *Vertex*, *Edge*, and *Attribute*, as well as a *ProvenanceController* singleton responsible for graph construction and updates at runtime. *ProvenanceContainer* provides graph lookup and traversal operations. This design preserves conceptual compatibility with PinG while aligning provenance management with Godot's scene tree and AutoLoad mechanisms.

A relevant characteristic of PinGODOT is that it remains fully functional without the automation layer: developers may create, manipulate, and export provenance

data manually through its core and interface components alone. In the context of this work, however, PinGODOT serves primarily as the structural and runtime foundation upon which PinGODOT Manager operates.

5.1. PinGODOT Manager

PinGODOT Manager is the editor-integrated extension that realizes the PinGA automation logic within Godot. Rather than requiring developers to insert tracking calls throughout gameplay scripts, the manager supports a declarative workflow in which provenance capture emerges from the game's configured execution. This design shifts a substantial part of the provenance instrumentation from source-code editing to engine-level configuration, reducing integration effort while preserving developer control over the semantic meaning of the collected elements.

The manager follows the same three-step logic introduced by PinGA. Candidate elements are first discovered from the Godot scene tree through node introspection. In this mapping, gameplay objects are represented as Entities, player-controlled or system-controlled actors may be represented as Agents, and relevant interactions, method calls, signal emissions, or state transitions are represented as Activities. Properties are stored as attributes of these provenance elements. The developer configures which discovered elements are semantically relevant, while the manager monitors the selected properties, methods, and signals at runtime and translates their changes into provenance updates sent to the PinGODOT core.

When a monitored interaction, method call, or property change is detected, PinGODOT Manager creates the corresponding provenance vertices and selects their relations through rules based on the source and target vertex types. These relations are generated automatically after monitoring is configured, but their semantic relevance still depends on the developer's selection and classification of gameplay elements.

Figure 2 illustrates this processing pipeline, from the selected gameplay node to provenance generation and export.

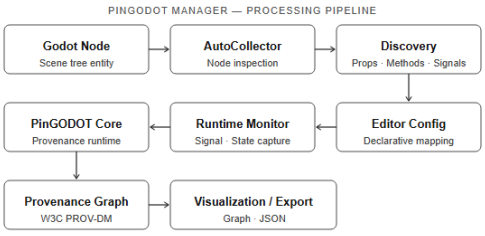

Figure 2. Automated provenance instrumentation pipeline implemented by PinGODOT Manager.

Godot Node represents the gameplay element selected as the entry point for provenance-oriented inspection, such as an object, character, or system node in the scene tree. **AutoCollector** is the main runtime component attached to selected gameplay nodes. It discovers trackable properties, methods, and signals through introspection, and monitors the elements enabled by the developer, allowing provenance capture to emerge from gameplay execution rather than from manually inserted tracking calls.

Discovery exposes the monitorable properties, methods, and signals of the configured target node, reducing the need to manually enumerate instrumentation targets in source code. **Editor Configuration** allows the developer to assign the target node a provenance role—Agent, Entity, or Activity—and select which properties, methods, and signals should be monitored. Selected properties are represented as attributes. **Runtime Monitoring** observes the configured elements during gameplay execution, capturing property changes, method calls, signal emissions, and interactions and converting them into provenance updates.

These updates are sent to the **PinGODOT Core**, which maintains graph consistency, registers provenance elements, and manages the runtime representation of the collected session. The data is then organized as a **Provenance Graph**, following the W3C PROV-DM model adopted by PinGODOT to represent causal relations among agents, activities, and entities. Finally, **Visualization / Export** is handled by *PinGManagerDock*, which supports JSON import/export and interactive graph visualization inside Godot, enabling recorded sessions to be inspected without external graph-processing tools.

6. Preliminary Evaluation

This section presents a preliminary evaluation of the Godot instantiation of PinGA, focusing on PinGODOT and PinGODOT Manager. The evaluation considers three main aspects: (i) integration effort, (ii) feasibility of the semi-automated workflow, and (iii) interpretability of the generated provenance graphs in a real game prototype. Rather than providing a large-scale performance study, this evaluation aims to verify whether the proposed workflow can be practically instantiated in a modern game engine and whether it can successfully generate useful provenance data during gameplay.

The integration procedure was analyzed as a feasibility-oriented measurement rather than as a controlled performance benchmark. The evaluation used *WILDLAND WARRIOR RUN*, a 2D runner prototype previously developed by one of the authors for an academic project on telemetry and gameplay data collection. The setup time was calculated from three individual integration trials, each performed by a different participant following the same procedure on the same game prototype. The procedure comprised copying the PinGODOT core folder into the target project, installing the PinGODOT Manager add-on in the `addons` directory, registering the required provenance controllers as AutoLoad singletons, enabling the plugin in the Godot editor, attaching `AutoCollector` nodes to selected gameplay elements, and selecting the properties, methods, and signals to be monitored through the editor interface. The exporter was instantiated on demand by the Manager and therefore did not require AutoLoad registration. Because the measurement involved only three participants and one prototype, the result may vary depending on project organization, developer familiarity with Godot, and the number of gameplay elements selected for monitoring.

These results indicate that the proposed workflow reduced scattered instrumentation calls but did not achieve full automation. In the *WILDLAND WARRIOR RUN* case study, integration took approximately seven minutes and required editor configuration, AutoLoad, AutoCollector, and 31 lines of manual code for initialization, semantic activity declaration, and export control. The 31 lines corresponded exclusively to these three tasks; element discovery and monitoring configuration were performed through the editor.

JSON input/output and graph visualization were supported.

After configuration, provenance collection and JSON export were successfully tested both in editor-executed sessions and in an exported Windows build of *WILD-LAND WARRIOR RUN*. Runtime collection and export are therefore supported in deployed builds, while graph import and visualization remain editor-integrated features.

The experiment confirmed that the workflow is operational end-to-end, covering editor-based configuration, runtime collection, JSON export, JSON import, and interactive graph visualization. This is particularly relevant because it demonstrates that the PinGA workflow can be incorporated into the standard development workflow of the engine without requiring external graph-processing tools or additional post-processing pipelines outside the editor environment.

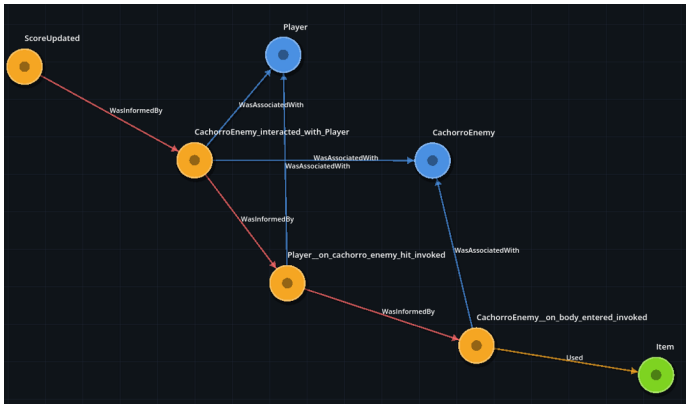

Figure 3. Partial provenance graph generated from the prototype and visualized inside the Godot editor.

The generated provenance graph contained vertices corresponding to agents, entities, and activities associated with relevant gameplay events such as game start, new session creation, player–enemy interactions, item collection, score updates, and defeat conditions. The resulting graph made it possible to inspect provenance relations connecting gameplay activities to their associated agents and to the entities they used or generated. In the evaluated prototype, activity-to-activity relations representing sequences such as item collection followed by score changes were explicitly produced by the semantic activity declarations included in the manual integration code, rather than inferred solely from temporal succession. Figure 3 shows a partial graph generated during the evaluation, illustrating how gameplay agents, entities, and activities are connected through provenance relations.

From an analytical perspective, these results indicate that the semi-automated workflow preserves the main objective of provenance-based telemetry: enabling developers to inspect not only what happened during gameplay, but also how recorded activities were related to gameplay agents and entities over time. In this sense, the graph was not merely a record of isolated events, but a structured representation of relationships that may support debugging, gameplay interpretation, and post-hoc behavioral analysis.

6.1. Discussion

Figure 4 contrasts manual provenance instrumentation in PinGU with the declarative automation workflow provided by PinGODOT Manager as an instantiation of PinGA.

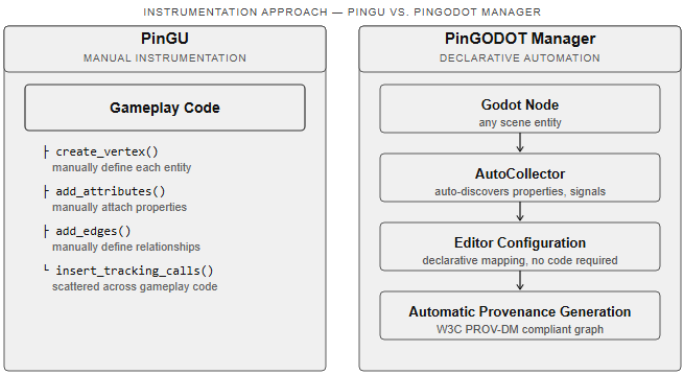

Figure 4. Comparison between manual provenance instrumentation and the declarative automation of PinGODOT Manager.

The comparison with PinGU indicates that PinGODOT Manager substantially reduces manual code-level instrumentation. In PinGU, the instrumentation effort grows with the number of events to be tracked, since each provenance vertex requires an explicit call that must be written and maintained in gameplay scripts. PinGODOT Manager replaces this event-by-event workflow with an editor-based declarative configuration: the developer attaches an `AutoCollector` to a scene node, assigns its target and provenance role, and selects the properties and methods to be monitored through the Inspector. The extension then discovers monitorable elements, detects configured changes and interactions, creates or updates provenance vertices, and selects edge relations through type-based rules. For monitored methods, the Manager also instruments the corresponding scripts automatically by inserting the signals and emissions required for runtime detection. Thus, the manual effort shifts from writing and maintaining one provenance call per event to configuring each relevant property or method once through the editor.

Nevertheless, complete automation has not yet been achieved. Initialization, selected semantic activities, and export control still require limited manual integration in the game code. Therefore, the current contribution should be understood as a hybrid approach that substantially reduces manual instrumentation while preserving developer control over semantically meaningful provenance collection.

Scalability is affected by the number of monitored elements and the length of the gameplay session. PinGODOT Manager mitigates data growth through selective monitoring, as each `AutoCollector` exposes individual flags for properties, methods, and signals. At the visualization level, repeated Activities can be grouped, the timeline exposes the cumulative evolution of the graph by displaying the provenance recorded up to a selected timestamp, and selecting a vertex highlights its direct provenance neighborhood. These operations affect only the visualization and preserve the original JSON data. Nevertheless, the evaluation did not quantify graph growth, memory consumption, runtime overhead, or interactive performance; therefore, it does not support scalability claims for large projects or long gameplay sessions. Overall, the results support the claim that PinGA can be feasibly instantiated in a modern game engine and used in small-scale

projects, prototypes, and educational contexts. In these settings, reducing instrumentation effort is particularly important, and provenance support can provide useful analytical feedback with relatively low integration cost.

6.2. Threats to Validity

This evaluation presents several limitations. First, the experiments were conducted on a single 2D prototype, which limits the generalizability of the results to larger-scale projects, especially 3D games with more complex runtime behavior, dynamic object creation, and more demanding scene hierarchies. Second, the observed setup time was measured in a controlled academic context and may vary depending on project organization, developer familiarity with Godot, and the complexity of the target game. Third, the automated discovery process relies primarily on syntactic introspection and does not guarantee that all monitored properties and methods are semantically relevant for provenance analysis. As a consequence, meaningful monitoring still depends in part on developer judgment during configuration. Fourth, the current workflow still requires limited manual coding to initialize provenance extraction, declare selected semantic activities, and control export operations, which means that the approach does not yet fully eliminate code-level integration. Finally, the present evaluation is qualitative and focused on feasibility rather than controlled quantitative comparison. Future work should therefore include larger-scale case studies, experiments with different genres and project sizes, and direct comparisons against manual instrumentation approaches in terms of integration effort, expressiveness, runtime overhead, and maintenance cost.

7. Conclusion

This paper presented **PinGA**, a semi-automated approach that combines automated discovery, developer-guided semantic configuration, and runtime monitoring to reduce manual provenance instrumentation. PinGA was instantiated in Godot through **PinGODOT**, which provides the runtime provenance infrastructure, and **PinGODOT Manager**, which integrates the automation workflow into the editor.

The preliminary evaluation in a 2D runner demonstrated end-to-end provenance collection, JSON export, in-editor import, and graph visualization, including runtime collection and export in a deployed Windows build. Compared with PinGU's event-by-event instrumentation, PinGODOT Manager shifts a substantial part of the manual code-level effort to declarative editor configuration. Nevertheless, initialization, selected semantic activities, and export control still require limited manual code. Future work includes improving semantic automation, evaluating larger and more diverse projects, measuring runtime and visualization scalability, and conducting controlled comparisons with manual instrumentation. The implementation, documentation, and example project are publicly available at `https://github.com/gems-uff/ping`..

References

Buneman, P., Khanna, S., e Tan, W.-C. (2001). Why and where: A characterization of data provenance. In *International Conference on Database Theory (ICDT)*, pages 316–330.

Drachen, A., Canossa, A., e Yannakakis, G. N. (2009). Player modeling using self-organization in Tomb Raider: Underworld. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games*, pages 1–8.

Drachen, A., Seif El-Nasr, M., e Canossa, A. (2013). Game analytics: The basics. In Seif El-Nasr, M., Drachen, A., e Canossa, A., editors, *Game Analytics: Maximizing the Value of Player Data*, pages 13–40. Springer, London.

Gagnon, D. J., Swanson, L., e Harpstead, E. (2024). Open game data: Defining a pipeline and standards for educational data mining and learning analytics with video game data. In *2024 IEEE Conference on Games (CoG)*, pages 1–8.

Gestwicki, P. (2021). Godot engine and checklist-based specifications: Revising a game programming class for asynchronous online teaching. *Journal of Computing Sciences in Colleges*, 37(4):30–40.

Holfeld, J. (2024). On the relevance of the godot engine in the indie game development industry. *arXiv preprint arXiv:2401.01909*.

Kohwalter, T. C., Clua, E. W. G., e Murta, L. G. P. (2012). Provenance in games. In *Proceedings of the Brazilian Symposium on Games and Digital Entertainment (SBGames)*, pages 162–171.

Kohwalter, T. C., Figueira, F. M. d. A., Serdeiro, E. A. d. L., Silva Junior, J. R., Murta, L. G. P., e Clua, E. W. G. (2018). Understanding game sessions through provenance. *Entertainment Computing*, 27:110–127.

Kohwalter, T. C., Murta, L. G. P., e Clua, E. W. G. (2017). Capturing game telemetry with provenance. In *Proceedings of the Brazilian Symposium on Games and Digital Entertainment (SBGames)*.

Lu, W., Griffin, J., Sadler, T. D., Laffey, J., e Goggins, S. P. (2023). Serious game analytics by design: Feature generation and selection using game telemetry and game metrics–toward predictive model construction. *Journal of Learning Analytics*, 10(1):168–188.

Mane, S. V. e Elmqvist, N. (2024). "wichita 1-1, fox three" – the role of 3d telemetry analysis in combat flight simulation. *Proceedings of the ACM on Human-Computer Interaction*, 8(CHI PLAY):349.

Moreau, L. (2010). The foundations for provenance on the web. *Foundations and Trends in Web Science*, 2(2–3):99–241.

Moreau, L. e Missier, P. (2013). PROV-DM: The PROV Data Model. Technical report, World Wide Web Consortium (W3C). W3C Recommendation.

Swanson, L. e Gagnon, D. J. (2024). An architecture for repeatable, large-scale educational game data analysis: Building on open game data. In *Serious Games*, volume 15259 of *Lecture Notes in Computer Science*, pages 41–55. Springer.

Thurler, L., Melo, S., Murta, L., Kohwalter, T., e Clua, E. (2025). Using provenance and replay for qualitative analysis of gameplay sessions. *Entertainment Computing*, 52:100778.