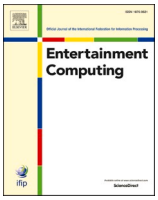
Entertainment Computing 52 (2025) 100777

Contents lists available at ScienceDirect

Entertainment Computing

journal homepage: www.elsevier.com/locate/entcom



Prov-DIFF: Play traces analysis through provenance differences

Troy Costa Kohwalter *, Leonardo Gresta Paulino Murta, Esteban Walter Gonzalez Clua

Departamento de Ciência da Computação, Universidade Federal Fluminense, Niterói, Rio de Janeiro, Brazil

ARTICLE INFO

Keywords:
Provenance
Game analytics
Session debugging
Play traces

ABSTRACT

A game session comprises a series of user decisions, inputs, and the execution of a strategy to reach specific goals. Tracking generated data of a game session is important for game analytics for developers and players. Game session data can be used for reproducibility, analysis of game traces, understanding player behavior, and improving the outcome in future sessions by learning from mistakes. However, game telemetry can rapidly lead to large amounts of data that can overwhelm the analyst's ability to analyze it, and it can be difficult to identify the reasons that might have caused a player to lose in that session. This paper proposes a provenance-based automatic debugging approach for game analytics. It identifies possible reasons and discrepancies that might have led a player to lose by contrasting their performance with other players. Our approach also proposes possible insights on how to improve the player's performance to reach the goal. We integrated our solution into the existing provenance visualization tool Prov Viewer. We provided an experimental study to demonstrate that our approach can identify probable causes that led the player to lose and propose changes to make it work in the next execution.

1. Introduction

Succeeding or losing in a game is the final consequence of a series of decisions, planning, and executing a strategy to overcome specific obstacles and achieve the objectives. Thus, an important problem in game analytics is understanding the reasons for certain outcomes and why some players failed while others managed to succeed. This analysis can also detect exploits and bugs in the game and discover winning strategies to overcome each challenge present in the game.

As such, game analytics has become an emerging field that is extremely popular and important for business intelligence in the game industry [1]. It provides a wealth of information for game designers, including feedback about design and gameplay mechanics, player experience, production performance, and even market reaction. Thus, the main goal of game analytics is to support the decision-making process at the operational, tactical, and strategic levels for game development.

However, game analytics still lacks standardization of key aspects and strategies, and, currently, the game industry adopts, in most cases, artisanal methods to understand the events of game sessions and determine the aspects that could have led players to fail certain goals, such as data crunching or game metrics [2], heat maps [3], and spatiotemporal clustering [4]. This analysis uses tracked game session data to identify factors or anomalies in the game that might have contributed to the outcome. Furthermore, existing practices adopted by the game industry for tracking game session data need to contain more information to determine probable causes for the reached outcomes.

Kohwalter et al. [5] proposed a novel approach named PinGU for capturing and storing provenance data from a game session based on the Provenance in Games conceptual framework [6]. Incorporating the PinGU framework in a game allows the developers to automatically capture and generate the game session provenance graph for analysis. A game provenance graph shows all actions, events, and agents and their relationships from a game session in an annotated graph that illustrates the temporal sequence of events and their causal relationships.

The provenance data collected during a game session is fundamental for understanding the mistakes made and reproducing the same results later. However, provenance data can be highly detailed and, depending on the game, can result in a huge quantity of tracked information, leading to provenance graphs that may have thousands of vertices. This wealth of information can overwhelm the developer's ability to analyze and understand the data, making identifying the reasons that may have caused a particular player to fail more complex and tedious when compared with the results from other players. Thus, in a follow-up work, Kohwalter et al. [7] proposed an approach for summarizing the provenance data to enhance the identification of sections or vertices that are

* Corresponding author at: Rua Passo da Pátria, 156 - Instituto de Computação - Sala 455 São Domingos, Niterói, RJ CEP: 24210-346, Brazil.
E-mail address: troy@ic.uff.br (T. Costa Kohwalter).

https://doi.org/10.1016/j.entcom.2024.100777
Received 22 April 2024; Accepted 15 June 2024
Available online 20 June 2024

different from their neighbors and might represent changes in the game state. However, that work is still focused on analyzing a single provenance graph at a time rather than comparing different game sessions to understand why some paths players took led to failure while others succeeded in reaching their goals. Therefore, Kohwalter et al. [8] also proposed the Provchastic approach that merges multiple provenance graphs into a single unified graph to statistically analyze the provenance data from multiple game sessions to predict outcomes using Markov Chains.

In this work, we propose a provenance graph comparison approach for game analytics capable of identifying possible reasons and discrepancies that might have led the player to lose a game session. Our proposed approach is inspired by spectra-based fault localization debugging [9], where we contrast the player's performance (*i.e.*, the failed trial) with the performance of other players that succeeded in reaching the desired goal (*i.e.*, the successful trial). Unlike Provchastic, which is focused on prediction, our approach, Prov-DIFF, provides the means for comparing provenance graphs from multiple sessions to determine the differences (diff) between graphs and understand the underlying reasons for the outcomes. This diff allows the designer, developers, and players to detect sections of the graph that differ from others. This can be used to discover the dissimilarities and the possible reasons behind each outcome by deriving the necessary actions or steps to reproduce the desired result. Furthermore, our approach proposes changes that can be made in the player's actions to improve his results in a future session.

We integrated our solution into the open-source provenance visualization tool Prov Viewer [10] and evaluated it through an experimental study regarding a projectile motion simulation. The experiment shows that the proposed approach can identify the reasons that could have led to a failure in the game. Moreover, our approach was able to propose changes to fix the failed trials in a subsequent execution by incorporating changes in the prospective provenance, reaching different levels of success, such as preserving a third of the original prospective provenance to reach 80 % accuracy in the projectile simulation or preserving almost half of the original decisions to reach 54 % accuracy. The more wrong decisions we make, the greater the chances of failing since all decisions in this simulation contribute to the simulation's goal of hitting the target.

The rest of the paper is organized as follows: Section II presents the related work in this area. Section III provides background knowledge related to provenance. Section IV presents our approach, and Section V presents the evaluation. Finally, Section VI provides final considerations and highlights future work.

2. Related work

Some studies identify patterns from data to determine what occurred or predict a game session's outcome through data mining, machine learning, and statistical analysis [11].

There are several approaches [12,13,14,15,16] that analyze game telemetry data to identify patterns and player behavior to predict outcomes, evaluating player performance during a match through predictions. Yang et al. [12] and Schubert et al. [13] approaches focus on combat patterns, or encounters, to predict the outcome of the match. Both approaches use captured game metrics to feed their predictive models, generating high-level predictive rules that do not consider contextual information, such as the sequence of executed actions. As such, it cannot reveal the dynamics of each combat. Only high-level factors tend to determine the game's outcome, such as hero level, gold, death, health, and damage during each encounter.

Similarly, Pobiedina et al. [14] also proposed an approach that uses statistical analysis of DotA 2 game data to identify factors that can increase a team's chances of winning, such as player role distribution in the team, friendship relations between players, leadership, and other player background information from previous matches, such as the amount of previously played and won matches, played time, and information about performance metrics in previous matches. With a similar focus on team effort, Eaton et al. [15] also proposed an approach using statistical analysis of League of Legends game data to identify impactful team members whose presence had a substantial effect on the outcome of the game and was related to victory. They use game metrics such as kill and assist count to determine the effectiveness of the players. However, neither approach is based on what happened during the match; only game metrics from player history and current match selection preferences are used to determine his winning probabilities.

Kleinman et al. [16] proposed a new approach for data analysis combining Sequence Analysis and Interactive Behavior Analytics [17] to aid analysts in examining player behavior using context information through the analysis of action sequences. Their approach adds behavioral labels to identified patterns to be used as a guide for the analysts to explore the game session data. However, this work focuses on understanding behavior rather than analyzing or inferring possible reasons for failure. It transfers all analytical processes to the analysts and provides no insights into player strategy.

Stafford et al. [18] also used an approach based on statistical analysis to identify skill development and improved player performance for the Destiny game through basic game metrics. They analyze historical player data to determine the learning curve. While this approach can determine if the player improved through the game sessions, it provides no insights into what influenced their performance.

Drenikow et al. [19] developed a tool for tracking and displaying player trajectories and in-game events to aid game designers in exploring the tracked data and identifying problems in their area designs. The Vixen tool provides the means for exploratory analysis. However, no game data analysis is done in their tool, and it transfers all the analysis responsibility to the cognitive ability of the end-user to interpret the data and extract insights.

Teng et al. [20] proposed the INSPECT system that leverages the player's playthrough data to generate a map of the player's behavior in the game. There are some limitations, such as the tool being unable to provide statistical analysis and determine the significance of the results. It can also only compare up to three sessions simultaneously. Lastly, the analysis is done visually by designers, developers, or players.

Green et al. [21] proposed a framework for analyzing game mechanics in terms of intrinsic and extrinsic rewards so it can be used as input for automated tutorial generation systems. It analyzes player behavior during the matches to determine his overall classification type and which type of intrinsic and extrinsic rewards would be more appropriate for that player. They aim to use these insights to customize automated content generation systems better. Sadly, these insights do not include strategies or actions the player needs to improve his performance.

All those cited works identify elements or provide predictions that might lead to successful (or winning the match) gameplay. However, they fail to provide insights into how the player strategies must be modified to improve performance. In other words, identify what went wrong and what should be done differently for more positive results in future sessions.

3. Provenance

Provenance is well understood in the context of art or digital libraries, where it refers to the documented history of an art object or the documentation of processes in a digital object's life cycle [22]. The *Open Provenance Model* (OPM) [23] was created during the *Provenance Challenge* [24], which is a collocated event of IPAW. Shortly after, another provenance model was developed, named PROV [25], which can be viewed as the successor of OPM. Both models aim to bring provenance concepts to digital data.

Provenance can be used for many purposes, including understanding how the data was collected to use it, determining the object's ownership, and deciding if the information is trustworthy. Mainly, it is used to show

the necessary steps to reproduce the results using an annotated causality graph, a directed acyclic graph enriched with annotations, also known as the provenance graph. According to Moreau *et al.* [23], a provenance graph is the "*record of a past or current execution and not a description of something that could happen in the future*." Similarly, the PROV model defines provenance as the "*information about entities, activities, and people involved in producing a piece of data, which can be used to assess its quality, reliability or trustworthiness*" [26]. This type of provenance is categorized as retrospective provenance.

€Three different vertices in the provenance graph, *Agents*, *Activities*, and *Entities*, can represent the following provenance information in PROV. The provenance graph uses shapes to distinguish the different types of vertices. The circle represents an entity, the square represents an activity, and the octagon represents an agent.

Entities represent physical or digital objects like items, equipment, or interactable objects. *Activities* represent the events or actions taken to change or interact with *entities* or *agents*. Lastly, an *agent* is a person, system, or entity with responsibilities, such as the player, enemies, and event managers. Furthermore, several *agents* can have responsibilities over the same *activity,* and a single *agent* can have responsibilities over several *activities*. *Agents* can also act on behalf of other *agents*, representing their interests when unavailable. These relations are some of the existing provenance relationships and are represented by edges in the graph.

Since the provenance graph captures causal dependencies between elements, it can be summarized using transitive rules. In previous work, we mapped these provenance concepts to the game's domain [6] to explore the advantages of registering a game session's provenance [27,28], showed how to capture game provenance data [5,29], summarize the provenance graph [7], and how to analyze multiple game sessions through unifying (merging) game provenance graphs [8]. The graph summarization combines similar sequential vertices from a graph to shrink it by removing redundant sequential data. Meanwhile, the graph unification combines (merges) two different provenance graphs into a single unified provenance graph.

Fig. 1 illustrates two examples of a provenance graph from two battles between the player (mage) and an enemy (orc) in a turn-based combat. The entity in the graph represents a healing potion that the player used during the battle. We have two agents: a mage and an orc. Each agent executes actions during their turn, such as attacking, casting spells, or drinking a potion. The edges represent the causal relationships between the activities, such as taking damage or an influence from a previous action. The health color legend shows that the activity's color is based on the health percentage. In the upper graph, the orc made a heavy attack and hit the mage, doing 20 health points of damage. Thus, in the graph, we have the activity that represents the heavy attack from the orc, connecting the orc agent, and an event "was hit" connected to the mage to represent that the attack hit the mage. Furthermore, we have an edge connecting the event with the attack, showing the activity's influence on the event, which was doing 20 hp damage. Moreover, the event is orange-colored, meaning that the mage's health is now between 30 % and 60 % after this attack, while the orc's attack is green-colored to represent that the orc is still in full health. In the mage's turn, he drank a healing potion to recover the damage he just took. Thus, he has an activity of "drank potion," connected to an entity "potion" to represent that the mage used an item, followed by an event of "was healed" with an edge connecting the two showing the amount of damage healed (20 hp). The distinction between an action and an event is encoded inside the vertex.

The combat continues until the orc kills the mage, so the mage loses the battle. The lower graph is similar, mostly changing the order of the mage's actions, which results in winning the battle against the orc.

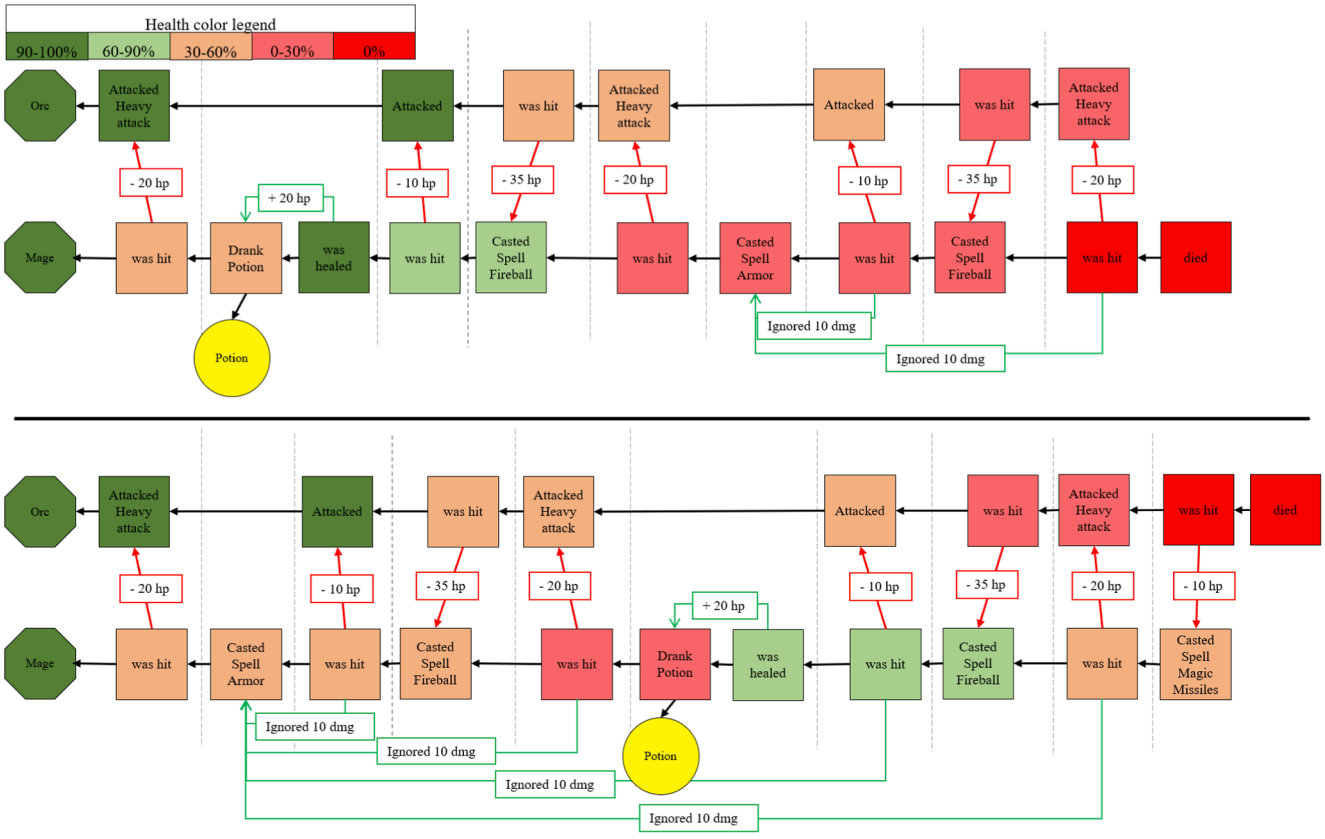


Fig. 1. Two game session provenance graph examples illustrate combat between the player (mage) and the enemy (orc). The player lost the battle in the upper graph and won in the bottom graph.

4. Prov-DIFF

This section discusses our approach to comparing provenance graphs generated by different game sessions. Our proposed approach is composed of three major phases: (1) **unified graph creation**, (2) **trial debugging**, and (3) **trial repair**, which are described in more detail in the following sections.

A. Unified Graph Creation

The first major phase, the **unified graph creation**, is responsible for creating the unified graph that is used by the second main phase to infer the probable causes that led to failure. We use the process of creating a unified graph proposed in a previous work [8] that requires four activities, as illustrated in Fig. 2: (1) a **matching heuristic** to match vertices from different graphs, (2) the definition of **vertex similarity**, (3) **vertex merge**, and (4) a **graph merge**. Note that the merge process is done by merging two graphs at a time; consequently, the matching heuristic uses only two graphs at a time.

The **matching heuristic** restricts the search space for vertex matching and avoids making a Cartesian product between vertices from both graphs. Furthermore, the heuristic decides how to compare vertices from different graphs. It always chooses two vertices (one from each graph) to pass them to the *Vertex Similarity* algorithm for comparison.

The **vertex similarity** algorithm, known as the distance metric function, compares two vertices to establish their similarity. The similarity value between two vertices ranges from 0 to 1, where 0 represents total mismatch (0 %), and 1 represents a perfect match (100 %). After establishing that two vertices are similar within an acceptable threshold, the next step is to merge them (**vertex merge**) to create a new vertex representing the two original vertices. This newly created vertex will belong to the unified graph. The vertex merge process creates a new vertex of the same type as the original vertices (*i.e.,* agent, activity, or entity), with all attributes from both vertices and their original values. In addition, it will show the minimum value, the maximum value, and the average value for each attribute. Furthermore, the merged vertex will have a new attribute showing the graphs to which it was used in the merger.

The **graph merge** activity is the last one to create a unified graph. This occurs only after the Matching Heuristic finishes matching vertices from both graphs. All the resulting merged vertices and vertices that were not matched are added to the unified graph, and finally, the edges are also added to the unified graph. However, the edges with an endpoint to any old vertices used for generating the merged vertex are updated to consider the newly merged vertex. Furthermore, the entire unification process records the provenance data on the origin of all vertices and edges in the newly created graph. This origin data is important for the debugging process.

B. Trial Debugging.

After generating the unified graph, comparing the differences between two or more graphs becomes trivial since each vertex contains information related to its origins (*i.e.*, the original graph it belonged to). Thus, we can filter vertices from a specific graph, highlight it inside the unified graph, and know how that specific graph differs from the other graphs. Furthermore, the original attribute values and their source are preserved when merging similar vertices.

The **trial debugging** uses the unified graph to detect a problem's causes and infer possible solutions through graph diffs. The comparison algorithm compares the *fail graph* (*i.e.*, the graph that had negative results or failed to achieve the goal) with all other graphs with positive results to find the discrepancies that could have explained the failure. Our algorithm searches in the unified graph for the *success graph* (*i.e.*, graph with positive results) with the shortest diff to the fail graph. It uses it as a baseline to determine the causes of the negative results through a vertex-by-vertex comparison. The vertices that only belong to the *fail graph* represent potential causes that might have led to the failure. Meanwhile, the vertices that appear only in the *success graph* are the suggested patch operation that contains the necessary knowledge to derive the required changes in the prospective provenance to reach the goal.

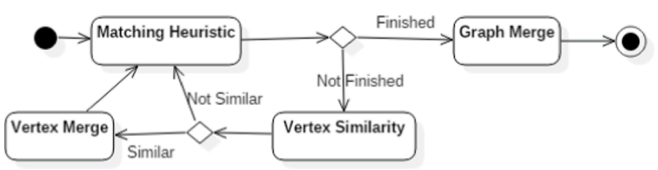


Fig. 2. Unified Graph Creation Process.

C. Trial Repair.

The *fail graph* might reach the desired goal if we replace the actions that appear only in the *fail graph* with the suggested actions in the **patch operation**, represented by the vertices that belong only to the *success graph*. In other words, we recommend to the player which actions they should stop doing and what actions they should start depending on the situation while preserving their overall playing style instead of simply mimicking someone else through granular suggestions. Nevertheless, it is also possible to make the *fail graph* identical to the *success graph* (mimic the behavior) by using a similarity function that considers vertices similar if they are 100 % equal in attributes and values. This could lead to an extremely large **patch operation**.

Fig. 3 illustrates a visual comparison between both graphs from Fig. 1, coloring in grey everything similar in both graphs. The upper graph is classified as a *fail graph* since the player died. The bottom graph is the *success graph* since the player won the battle. Fig. 4 shows the resulting unified graph from those two provenance graphs and colors the information common in both graphs in gray. All actions, influences, and events colored in grey were considered "correct" by the Prov-Diff algorithm because they appear in both graphs. *Red vertices* represent actions and events that appeared in the *fail graph* but not in the *success graph*. These vertices might encode the reason for not reaching the goal. *Green vertices* represent actions and events that appear in the *success graph* but do not appear in the *fail graph*. These vertices are used to suggest the **patch operation** to make *the failed graph* reach the goal after incorporating the respective changes in the prospective provenance. Fig. 5 shows the actions that need to be replaced (in red) and the suggested actions that need to be performed instead (in green) to improve the result, which, in this example, is winning the battle. Notice that the moment they are performed is important relative to other actions. This sequence of actions relative to other actions is encoded in the graph through the edges, while temporal information is inside the vertices. White nodes represent events that are a consequence of the actions, not the actions themselves.

5. Evaluation

In this section, we assess our proposed approach for provenance graphs to detect the possible causes of failure by comparing the *fail graph* with the *success graph* and suggesting a probable fix. We evaluate our approach through the following research questions:

RQ1: Can the proposed approach correctly detect the causes of the failure?

RQ2: Does preserving multiple segments of the fail graph impact the results from our algorithm?

We answer these research questions through three dependent variables: *accuracy*, *retention*, and *harmonic mean*. The *accuracy* metric tells us how many times our algorithm correctly predicted the probable causes that led to failing the goal. This is measured by applying the **patch operation** on the prospective provenance from the resulting *fail graph* and verifying if the included changes effectively reached the goal. If the proposed changes proved successful in future runs, our algorithm correctly predicted the causes of the failure. Therefore, the higher the *accuracy*, the better.

The *retention* metric tells us how many vertices remained unaffected

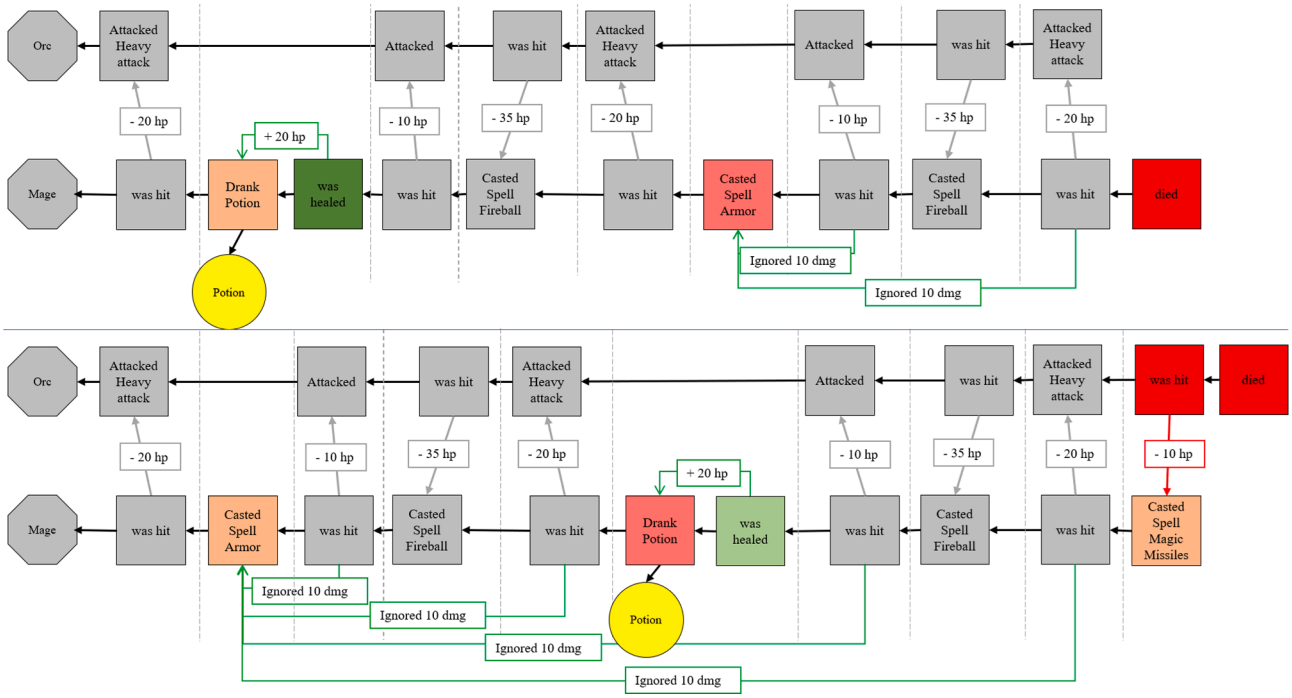


Fig. 3. Side-by-side comparison diff. The upper graph is the *fail graph* (player died), and the lower graph is the *success graph* (player won).

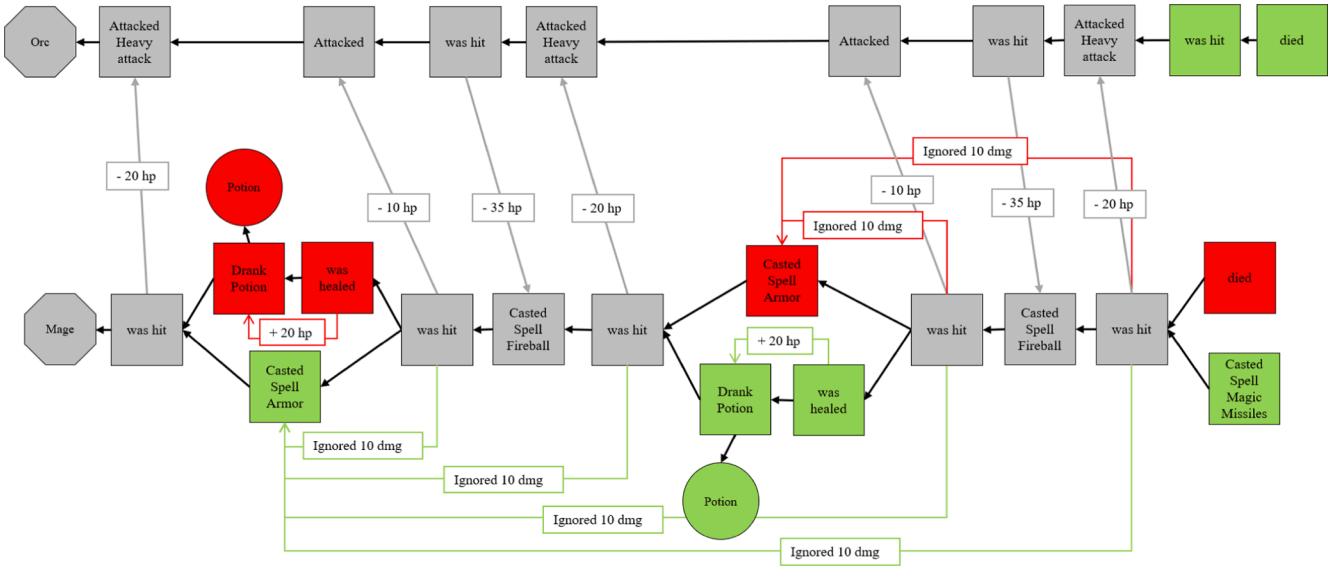


Fig. 4. Comparison visualization using the unified graph.

or unaltered by the **patch operation**. This metric measures how much of the *fail graph* was preserved and allows us to compare the algorithm accuracy based on the number of changes in the **patch operation**. Reaching 100 % *accuracy* is easy if the *retention* rate is near zero since it translates to changing the majority or all the necessary vertices from the *fail graph* to be an exact clone of one of the *success graphs*. However, this is only sometimes desirable or possible because sometimes we want to preserve a good portion of the *fail graph* for a particular reason, such as preserving the player's overall behavior. For example, if we contrast a casual player's performance with a professional one, the algorithm changes everything. The casual player would need to figure out where to invest in improving his future performance. Therefore, the ideal would be high *accuracy* and high *retention* values, representing the minimum patch on the *fail graph* that transforms it into a *success graph*.

However, the increase of one metric normally tends to decrease the others. Thus, the last metric we use is the *harmonic mean,* which tells us the algorithm's overall performance based on the compromise of accuracy and retention metrics.

A. Materials and Method.

This experiment was executed using a shooting competition game simulation that uses projectile motion physics to hit a target. We evaluated the accuracy of Prov-DIFF in detecting the probable causes that led to missing the target.

The game simulation has nine configurable parameters: (1) bullet mass, (2) air density based on temperature, (3) air density based on altitude, (4) air drag, (5) initial X position, (6) initial Y position, (7) bullet speed on X-axis, (8) bullet speed on Y axis, and (9) target position. Moreover, it also uses a constant for gravity. The physics behind this

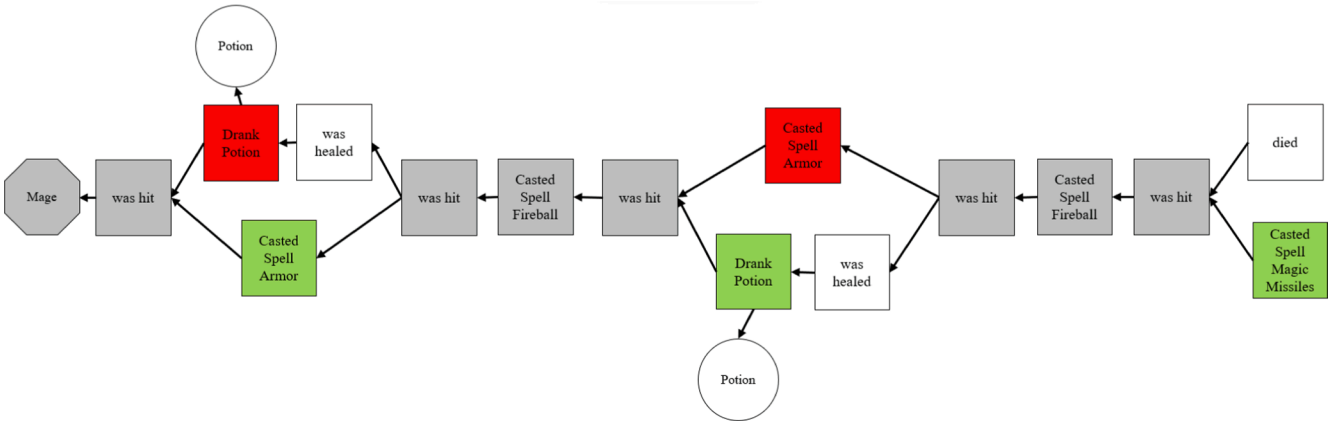


Fig. 5. Patch suggestion to improve the result.

simulation is described in Equation 1, where ρ is the air density, C_d is the drag coefficient, and A is the cross-sectional area of the projectile. The simulation goal is for the shooter to hit the target.

Equation 1: Projectile motion equations

$$\ddot{x} = -\beta \dot{x} \sqrt{\dot{x}^2 + \dot{y}^2}$$

$$\ddot{y} = -g - \beta \dot{y} \sqrt{\dot{x}^2 + \dot{y}^2}$$

$$\beta = \frac{\alpha}{m}$$

$$\alpha = \frac{\rho C_d A}{2}$$

We generated a computer simulation of a shooting competition with 15 participants. The competition comprises 20 rounds; in each round, all participants have one shot to hit the target. The target position changes after each round. Each shot generates a provenance graph with the parameters of the projectile motion simulation. The input parameters were generated randomly using a Gaussian distribution to simulate different players participating in the competition. We consulted real data values to select the mean for the Gaussian function and generate the sigma for the Gaussian distribution. As a result, all input variables were continuous. Furthermore, for this experiment, we only consider the vertices that represent decisions related to the nine input parameters since those are the only decisions that can be made in the simulation by the player or the match configuration (everything else is a consequence of those decisions).

Lastly, the results from our comparison algorithm are related to the unified graph, which requires a *similarity threshold* to determine if two graphs are similar and calculate the diffs. All variables in this experiment belong to the continuous space, meaning there will never be two equal values in the entire dataset. Moreover, the *similarity threshold* is directly related to the *retention* metric because a value from the *fail graph* can be considered similar to its counterpart from the *success graph* even though they are not equal, thus preserving the *fail graph* vertex. Therefore, the **patch operation** will not change the vertex since it is equivalent to the one from the *success graph*.

As such, the experiment execution plan was divided into five stages: (1) generate the dataset, (2) create different similarity thresholds to analyze the impact of the *retention* vs. *accuracy*, (3) generate the unified graph for each similarity threshold from stage 2, (4) execute the experiment using the unified graphs from stage 3, and (5) analyze the results. The simulation resulted in 300 graphs in the first stage since it had 15 participants and 20 rounds, totaling 300 shots. Of these 300 generated graphs, only 16 hit the target, equivalent to 5.33 % of the shots. Prov-DIFF requires at least one *success graph*, similar to spectra-based fault localization debugging, and this restriction was satisfied in the generated dataset.

The second stage is responsible for generating the *similarity threshold* for the experiment. Thus, to evaluate the impact of retaining multiple elements from the *fail graph*, we created ten different *similarity thresholds* that result in different retention rates. Each *similarity threshold* uses the same factor of standard deviation (sigma) to define the similarity threshold for each input variable. The difference between the *similarity thresholds* is the factor used for the thresholds. The generated *similarity threshold* is: (1) 0-Sigma, (2) 0.25-Sigma, (3) 0.5-Sigma, (4) 0.75-Sigma, (5) 1.0-Sigma, (6) 1.25-Sigma, (7) 1.5-Sigma, (8) 1.75-Sigma, (9) 2.0-Sigma, (10) 3.0-Sigma. Thus, the first metric (0-Sigma) will only consider two vertices to be similar if their attributes' values are within zero standard deviations apart or, in other words, if their numeric values are the same. We did not add more points between 2.0-Sigma and 3.0-Sigma due to the rule "68–95-99.7", which states that, for a normal distribution, 68 % of the values fall between one Sigma around the mean, 95 % fall between two Sigma around the mean, and 99.7 % fall between three Sigma around the mean. Thus, the difference between the zones for 2-Sigma and 3-Sigma is only 4.7 %, which is too small to generate any significant impact on the result. Nevertheless, we kept 2-sigma and 3-sigma in the experiment to evaluate whether the difference between these metrics is significant.

The third stage creates the unified graphs using the *similarity threshold* from the second stage and the 300 graphs from the first stage. Thus, at the end of this stage, we had ten different unified graphs that represented different *similarity thresholds* when applied over the same 300 graphs. Table 1 shows data from those ten unified graphs related to the total number of vertices, the number of vertices that only appeared on *success graphs*, the number of vertices that only appeared on *fail graphs*, and the number of vertices that appeared on both.

Finally, we executed the experiment using the unified graphs from the third stage. We calculated each unified graph's accuracy, retention,

Table 1
Unified graph comparisons for different similarity thresholds.

Similarity Threshold	# Vertices	# Success Vertices	# Fail Vertices	# Common Vertices
0.00-Sigma	2420	128	2282	10
0.25-Sigma	142	0	78	64
0.50-Sigma	87	0	38	49
0.75-Sigma	66	0	27	39
1.00-Sigma	58	0	21	37
1.25-Sigma	54	0	17	37
1.50-Sigma	53	0	18	35
1.75-Sigma	50	0	11	39
2.00-Sigma	40	0	13	27
3.00-Sigma	18	0	2	16

and harmonic mean metrics. This process was done by applying the **patch operation** in each graph that had yet to reach the goal for each one of the unified graphs (284 graphs from 300). Then, we ran the projectile motion simulation in each patched graph to determine if the modifications were sufficient to allow the shot to hit the target. *Accuracy* is calculated by dividing the number of graphs that succeeded after the patch by the total number of graphs that needed to be corrected (*i.e.*, 284). Using the 0.5-Sigma example, we had 154 successful corrections from 284, approximately 54 %. *Retention* is calculated using the average retention of the 284 graphs, which is the minimum diff size for each graph since the diff size reflects the number of suggested changes. In the 0.5-Sigma, the average number of changes in all 284 graphs was 3.8 from the nine configurable parameters, which is a 42 % *retention* rate. The *harmonic mean* is calculated based on the *accuracy* and *retention*, which in the 0.5-Sigma example results in 48 %.

B. Results and Discussion

Fig. 6 shows the results obtained from our evaluation for each similarity threshold used to generate the unified graph. Regardless of whether the patch worked, the retention rate was calculated for all graphs.

These results show that our comparison algorithm can achieve a 100 % accuracy rate, answering RQ1. However, this only occurs when using the 0-sigma similarity, which has the lowest retention rate of 5 %. This means that the algorithm discarded all elements from the failed graph that were not equal to the other graph and replaced them with elements from the graph that reached the goal due to the zero-similarity threshold in a continuous domain, resulting in a graph equal to the successful one.

The results from other *similarity thresholds* show that the *accuracy* and *retention* metrics are inversely proportional, related to RQ2: *accuracy* decreases as the *retention* rate increases. This also sounds natural since if we preserve the failed graph, then the goal will not be reached. Furthermore, the more elements we preserve from the failed graph, the lower the chances are of identifying the reasons and fixing them since the cause of the failure can be broader than what can be changed by the algorithm. Moreover, in this game, all factors contribute to the projectile trajectory. This allows the user to choose the desired *retention* rate of the analyzed graph at the cost of losing the *accuracy* of the recommendation based on the patch size. In other words, the user chooses how much the algorithm will preserve his overall playing style at the cost of reducing the accuracy of the recommendation.

Thus, it is up to the user to define whether *accuracy* or *retention* is more important. However, looking at the obtained results, it is not recommended to have a *retention* rate greater than 55 % because otherwise, the average *accuracy* rate drops from 39 % to 18 %. The Harmonic mean metric can be used in cases where the user is after the algorithm's overall performance based on the compromise of *accuracy* and *retention* metrics. Looking at the Harmonic mean metric, the optimal *similarity threshold* would be 0.5-Sigma, resulting in a 54 % *accuracy* and a 42 % *retention* rate. However, we can reach a much higher accuracy (80 %) by preserving 30 % of the original failed decisions with only a 12 % loss in retention rate. Moreover, although 54 % or even 80 % accuracy can be considered low in some situations, it is important to point out that in this simulation, from the observed data, the chances of scoring a hit in the target, which translates into a successful shot, is only 5.33 %. Thus, the 54 % accuracy of our approach when retaining 42 % of the original decisions is still much higher (almost ten times higher) than simply making another random shot. The player would need to follow the recommended moves from our algorithm (*i.e.*, the proposed changes in the **patch operation**) to improve his odds in the next session.

Furthermore, the results show that our algorithm can work with a flexible definition of similarity. However, this flexibility for similarity definition directly impacts our algorithm's accuracy, as demonstrated by the experiment. This, from a logical point of view, makes sense since we are broadening the definition of similar values and, as a result, increasing the acceptable error margins in the interpretation of what is considered similar objects, which, in turn, can lead to errors in cases where two objects are similar even when they are completely different.

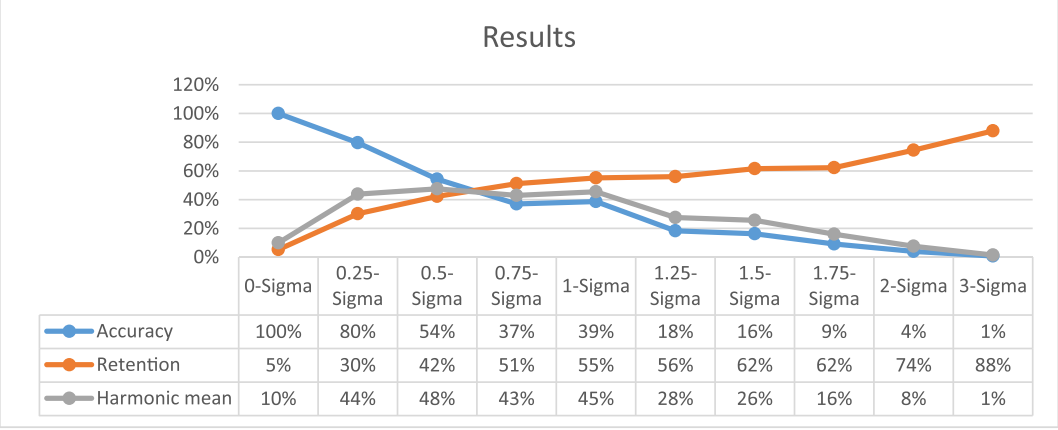
Lastly, our approach considers only the closest correct execution to derive the answer. A possible improvement would be to compare the faulty execution with multiple correct executions, instead of only the closest one, to my successful patterns and derive an answer with fewer changes. Another possibility would be to merge all correct executions and use this merged graph as the *success graph*.

C. Threats to Validity.

We identified internal and external threats that may influence the results. About internal validity, we had to generate all 300 provenance graphs to run the experiment. We used a Gaussian distribution for each configurable parameter to generate acceptable random values within the parameter domain for all the 300 different graphs to minimize the effect of using completely random values that might not reflect real gameplay data. Regarding external validity, we mitigated sample bias by randomly generating 300 different graphs and having a set with more than one graph that reached the goal. However, we only did one case study, which might threaten the generalization of our results.

6. Conclusion

An important problem is understanding the reasons for certain outcomes and why some players failed to achieve the final goal while others managed to reach it. Thus, we proposed a provenance approach to understand how an outcome was reached. We developed a provenance graph merge and comparison approach in the context of digital game sessions capable of identifying possible reasons and discrepancies in a provenance graph that might have led a player to fail to reach the goal.



	0-Sigma	0.25-Sigma	0.5-Sigma	0.75-Sigma	1-Sigma	1.25-Sigma	1.5-Sigma	1.75-Sigma	2-Sigma	3-Sigma
Accuracy	100%	80%	54%	37%	39%	18%	16%	9%	4%	1%
Retention	5%	30%	42%	51%	55%	56%	62%	62%	74%	88%
Harmonic mean	10%	44%	48%	43%	45%	28%	26%	16%	8%	1%

Fig. 6. Evaluation results showing Accuracy, Retention, and harmonic mean metrics for each similarity *threshold* used.

To do so, we contrast the provenance of the failed game session with the combined provenance of all successful game sessions.

Our approach can detect the decisions or underlying issues that could have led to failure by comparing them with another provenance graph known to reach the goal, similar to spectra-based fault localization debugging. The results over a specific shooting game showed that our approach reached 80 % *accuracy* with a 30 % *retention* rate, meaning that it could fix 80 % of the failed trials while preserving 30 % of the original (fail) graph. However, the algorithm can reach 100 % *accuracy* with a low *retention* rate (close to 5 %). Like the spectra-based fault localization debugging technique, the size and diversity of the sample that contains correct executions impact the algorithm results. Furthermore, the chances of hitting the target with a random shot are only 5.33 %, and our approach managed to increase this chance drastically (54 % *accuracy* with a 42 % *retention* rate and 80 % *accuracy* with a 30 % *retention* rate), as shown during the patch operation and the re-execution of the simulation.

While the main application of provenance in this work is for games, the concepts apply to other domains and might be useful to support advanced forms of analysis. When used outside the games domain, our proposed approach can help debug experimental trials to determine why a specific trial failed while another had positive results and derive a possible fix from that knowledge. Thus, the proposed concepts could be applied to scientific experiments to debug the experiment to identify issues and better understand the obtained results by exporting the provenance graphs using the PROV-N notation or another compatible with *Prov Viewer*. Different AI learning techniques can use the Ou approach to learn from their mistakes and observations from other players or previous session data to adapt their actions or mimic player behavior. Furthermore, our model could also be used as a stepping stone for applying different AI learning methods and techniques to make inferences about the player's actions and behavior, which could be used for automatic game balancing, calibration, and content adaption.

A limitation of our comparison approach is that it requires at least one graph to reach the goal. Otherwise, it would be unable to narrow down the possible reasons for failure. Furthermore, the algorithm's effectiveness is linked to the unified graph and the definition of the *similarity threshold*.

Future works related to comparing provenance graphs include finding good patterns from graphs that reached their goals to improve the chances of reaching the same goal in future iterations. Similarly, another approach could be detecting bad patterns that should be avoided or using statistical analysis or artificial intelligence techniques using data from previous executions to infer future outcomes based on an ongoing session or hypothetical situations. We also plan to perform a deeper evaluation using an open-source game. Lastly, this work can be used with Provchastic to recommend traces with better chances of success and improve the player's performance even further.

CRediT authorship contribution statement

Troy Costa Kohwalter: Conceptualization, Data curation, Formal analysis, Methodology, Project administration, Software, Validation, Visualization, Writing – original draft. **Leonardo Gresta Paulino Murta:** Funding acquisition, Supervision, Writing – review & editing. **Esteban Walter Gonzalez Clua:** Funding acquisition, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

We thank CNPq, FAPERJ, and CAPES for their financial support.

References

[1] M. El-Nasr, A. Drachen, and A. Canossa, Eds., *Game Analytics - Maximizing the Value of Player Data*. In: Springer Science & Business Media, 2013. Accessed: Feb. 12, 2015. [Online]. Available: http://www.springer.com/computer/hci/book/978-1-4471-4768-8.

[2] A. Drachen, A. Canossa, Towards Gameplay Analysis via Gameplay Metrics, Int. Mindtrek Conf. Everyday Life Ubiquitous Era (2009) 202–209, https://doi.org/10.1145/1621841.1621878.

[3] A. Drachen, A. Canossa, Analyzing spatial user behavior in computer games using geographic information systems, MindTrek Conf. Everyday Life Ubiquitous Era (2009) 182–189, https://doi.org/10.1145/1621841.1621875.

[4] C. Bauckhage, R. Sifa, A. Drachen, C. Thurau, F. Hadiji, "Beyond heatmaps: Spatio-temporal clustering using behavior-based partitioning of game levels," in IEEE Conference on Computational Intelligence and Games, Aug. 2014, pp. 1–8. doi: 10.1109/CIG.2014.6932865.

[5] T.C. Kohwalter, L.G.P. Murta, E.W.G. Clua, Capturing Game Telemetry with Provenance, Braz. Symp. Games Digit. Entertain, SBGAMES, 2017.

[6] T. Kohwalter, E. Clua, L. Murta, "Provenance in Games," Braz. Symp. Games Digit. Entertain. SBGAMES, pp. 162–171, 2012.

[7] T. Kohwalter, L. Murta, E. Clua, Filtering irrelevant sequential data out of game session telemetry though similarity collapses, Future Gener. Comput. Syst. 84 (Jul. 2018) 108–122, https://doi.org/10.1016/j.future.2018.03.004.

[8] T. C. Kohwalter, L. G. P. Murta, and E. W. G. Clua, "Provchastic: Understanding and Predicting Game Events Using Provenance," in Entertainment Computing – ICEC 2020, N. J. Nunes, L. Ma, M. Wang, N. Correia, and Z. Pan, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 90–103. doi: 10.1007/978-3-030-65736-9_7.

[9] M.J. Harrold, G. Rothermel, K. Sayre, R. Wu, L. Yi, An empirical investigation of the relationship between spectra differences and regression faults, Softw. Test. Verification Reliab. 10 (3) (2000) 171–194, https://doi.org/10.1002/1099-1689(200009)10:3<171::AID-STVR209>3.0.CO;2-J.

[10] T. Kohwalter, T. Oliveira, J. Freire, E. Clua, L. Murta, "Prov Viewer: A Graph-Based Visualization Tool for Interactive Exploration of Provenance Data," in Proceedings of the 6th International Provenance and Annotation Workshop on Provenance and Annotation of Data and Processes - Volume 9672, in IPAW 2016. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 71–82. doi: 10.1007/978-3-319-40593-3_6.

[11] D. Hooshyar, M. Yousefi, H. Lim, Data-Driven Approaches to Game Player Modeling: A Systematic Literature Review, ACM Comput. Surv. 5 (6) (2018) 90:1–90:19, https://doi.org/10.1145/3145814.

[12] P. Yang, B. Harrison, D.L. Roberts, "Identifying Patterns in Combat that are Predictive of Success in MOBA Games," *Found. Digit. Games FDG*, p. 8, 2014.

[13] M. Schubert, A. Drachen, T. Mahlmann, "Esports Analytics Through Encounter Detection," presented at the MIT Sloan Sports Analytics Conference, MIT Sloan, 2016. Accessed: May 11, 2021. [Online]. Available: http://lup.lub.lu.se/record/8569749.

[14] N. Pobiedina, J. Neidhardt, M. del C. Calatrava Moreno, L. Grad-Gyenge, and H. Werthner, "On Successful Team Formation: Statistical Analysis of a Multiplayer Online Game," in 2013 IEEE 15th Conference on Business Informatics, Jul. 2013, pp. 55–62. doi: 10.1109/CBI.2013.17.

[15] J.A. Eaton, D.J. Mendonça, M.-D.-D. Sangster, Attack, Damage and Carry: Role Familiarity and Team Performance in League of Legends, Proc. Hum. Factors Ergon. Soc. Annu. Meet. 62 (1) (Sep. 2018) 130–134, https://doi.org/10.1177/1541931218621030.

[16] E. Kleinman *et al.*, "'And then they died': Using Action Sequences for Data Driven, Context Aware Gameplay Analysis," in International Conference on the Foundations of Digital Games, in FDG '20. New York, NY, USA: Association for Computing Machinery, Sep. 2020, pp. 1–12. doi: 10.1145/3402942.3402962.

[17] S. Ahmad, A. Bryant, E. Kleinman, Z. Teng, T.-H. D. Nguyen, M. Seif El-Nasr, "Modeling Individual and Team Behavior through Spatio-temporal Analysis," in Proceedings of the Annual Symposium on Computer-Human Interaction in Play, in CHI PLAY '19. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 601–612. doi: 10.1145/3311350.3347188.

[18] T. Stafford, S. Devlin, R. Sifa, and A. Drachen, "Exploration and Skill Acquisition in a Major Online Game," The 39th Annual Meeting of the Cognitive Science Society (CogSci). Accessed: May 11, 2021. [Online]. Available: http://eprints.whiterose.ac.uk/118051/.

[19] B. Drenikow, P. Mirza-Babaei, "Vixen: interactive visualization of gameplay experiences," in Proceedings of the 12th International Conference on the Foundations of Digital Games, in FDG '17. New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 1–10. doi: 10.1145/3102071.3102089.

[20] Z. Teng, J. Pfau, S. S. Maram, M. Seif El-Nasr, "Player Segmentation with INSPECT: Revealing Systematic Behavior Differences within MMORPG and Educational Game Case Studies," in Extended Abstracts of the 2022 Annual Symposium on

Computer-Human Interaction in Play, in CHI PLAY '22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 87–92. doi: 10.1145/3505270.3558340.

[21] M. C. Green, A. Khalifa, R. Canaan, P. Bontrager, J. Togelius, "Game Mechanic Alignment Theory," in Proceedings of the 16th International Conference on the Foundations of Digital Games, in FDG '21. New York, NY, USA: Association for Computing Machinery, Outubro 2021, pp. 1–11. doi: 10.1145/3472538.3472571.

[22] PREMIS Working Group, "Data Dictionary for Preservation Metadata", Implementation Strategies (PREMIS), OCLC Online Computer Library Center & Research Libraries Group, Final report, 2005.

[23] L. Moreau, et al., The Open Provenance Model core specification (v1.1), Future Gener. Comput. Syst. 27 (6) (2007) 743–756, https://doi.org/10.1016/j.future.2010.07.005.

[24] S. Miles, J. Heasley, A. Szalay, L. Moreau, and P. Groth, "Provenance Challenge WIKI." Accessed: Mar. 26, 2013. [Online]. Available: http://twiki.ipaw.info/bin/view/Challenge/.

[25] L. Moreau and P. Missier, "PROV-DM: The PROV Data Model." Accessed: Mar. 21, 2013. [Online]. Available: http://www.w3.org/TR/prov-dm/.

[26] T. D. Nies, J. Cheney, P. Missier, L. Moreau, "Constraints of the PROV Data Model." Accessed: Mar. 21, 2013. [Online]. Available: http://www.w3.org/TR/prov-constraints/.

[27] T. Kohwalter, E. Clua, L. Murta, Game Flux Analysis with Provenance, Adv. Comput. Entertain. ACE (2013) 320–331.

[28] T. Kohwalter, E. Clua, and L. Murta, "Reinforcing Software Engineering Learning through Provenance," 2014 Braz. Symp. Softw. Eng. SBES, pp. 131–140, Sep. 2014, doi: 10.1109/SBES.2014.16.

[29] T. Costa Kohwalter, F.M. de Azeredo Figueira, E.A. de Lima Serdeiro, J.R. da Silva Junior, L. Gresta Paulino Murta, E. Walter Gonzalez Clua, Understanding game sessions through provenance, Entertain. Comput. 27 (2018) 110–127, https://doi.org/10.1016/j.entcom.2018.05.001.