Entertainment Computing 27 (2018) 110–127

Contents lists available at ScienceDirect

Entertainment Computing

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

Understanding game sessions through provenance

Troy Costa Kohwalter[a,*,1], Felipe Machado de Azeredo Figueira[b,2], Eduardo Assis de Lima Serdeiro[b,2], Jose Ricardo da Silva Junior[b,2], Leonardo Gresta Paulino Murta[a,1], Esteban Walter Gonzalez Clua[a,1]

[a] Universidade Federal Fluminense, Brazil
[b] Instituto Federal do Rio de Janeiro, Brazil

ARTICLE INFO

Keywords:
Game analytics
Tracked game data
Provenance graph

ABSTRACT

The outcome of a gameplay session is derived from a series of events, decisions, and interactions made during the game. Many techniques have been developed by the game industry to understand a gameplay session. A successful technique is game analytics, which aims at understanding behavior patterns to improve game quality. However, current methods are not sufficient to capture underlying cause-and-effect relationships that occur during a gameplay session, which would allow designers to better identify possible mistakes in the mechanics or fine-tune their game. Recently, it was proposed a conceptual framework based on provenance to capture these relationships. In this paper, we present a concrete framework to capture provenance data, allowing developers to add provenance gathering capabilities to their games. We instantiated our framework in two games, showing how it can be used in practice, and we developed a new game to demonstrate how provenance could be employed in early stages of game development to assist balancing the difficulty. We conducted an experiment with twelve volunteers and used the gathered provenance data to answer designers' frequent questions when trying to understand game sessions and balancing the difficulty of their games. This supports the relevance of collecting provenance data from games.

1. Introduction

The analysis of tracked game data, also known as game telemetry, has become an important stage of game design and production in the last few years [1]. This gathered data brings relevant information and possibilities, such as measuring the game stability [2], dynamically adjusting the difficulty of the game [3], performing behavioral analysis [4], understanding common behaviors [5], improving the monetization process [1], and balancing the game experience [6]. Moreover, game telemetry allows game developers to collect player interactions in the game inconspicuously over extended time periods, during production and after deployment.

Tracking game data and making it understandable is challenging due to the complexity of the games, leading to huge amounts of information. Additionally, deciding which information should be tracked and recorded is another challenge. One of the most common types of telemetry data is through states changes [7–9]. Even though state data is easier to examine, they typically lack contextual information and provides only a high-level view of what happened in the game. In contrast, telemetry data that captures events [10,11], can provide more low-level and fine-grained information, capturing and describing the player activity and relating it more closely to the game session. Furthermore, since the data is collected at fine-grain, developers can use aggregating techniques to summarize the data by giving an overview of the game sessions and only digging through the fine-grained data when necessary.

However, no known approaches for game analytics take into consideration the cause-and-effect relationships between events during a game session, which may be an important factor for determining the reasons that led to a certain outcome. In a recent work, Kohwalter et al. [12] introduced the usage of digital provenance [3] in games in order to detect these cause-and-effect relationships through a conceptual framework, named *Provenance in Games* (PinG), that can collect information during a game session and maps the data to provenance terms,

* Corresponding author.
E-mail addresses: tkohwalter@ic.uff.br (T. Costa Kohwalter), felchado@gmail.com (F.M. de Azeredo Figueira), eduardoassislima@gmail.com (E.A. de Lima Serdeiro), jose.junior@ifrj.edu.br (J.R. da Silva Junior), leomurta@ic.uff.br (L. Gresta Paulino Murta), esteban@ic.uff.br (E. Walter Gonzalez Clua).
[1] Address: Av. Gal. Milton Tavares de Souza, s/nº, Niterói, RJ, Brazil.
[2] Address: Av. Maria Luiza, s/nº, Eng. Paulo de Frontin, RJ, Brazil.
[3] Provenance refers to the documented history of an object's life cycle and is generally used in the context of art, digital data, and science [13].

https://doi.org/10.1016/j.entcom.2018.05.001
Received 1 March 2018; Received in revised form 26 April 2018; Accepted 6 May 2018

providing the means for a post-game analysis. That conceptual framework was manually instantiated over a game named SDM [14], which focuses on teaching Software Engineering concepts. The provenance support in that game allowed for a broader range of analysis by using collected provenance information to generate provenance graphs [15]. Even more recently, Kohwalter et al. [16] also demonstrated the benefits of using their PinG approach during game analysis of serious games, helping students to understand the underlying reasons for an outcome.

In this paper, we present a concrete framework for capturing provenance data for the game engine Unity, allowing developers to add provenance gathering capabilities to their games. We detail how our proposed framework can be instantiated in an existing game and show how the generated provenance graph can be visualized using the provenance visualization tool *Prov Viewer* [17], which is a visualization tool that supports multiple features for visual data analysis, including spatial-referencing the graph in the game level map.

We also provide an evaluation on how our framework can be used for the analysis of the cause-and-effect relationships by instantiating it over two open source games released by the Unity team. Additionally, we also demonstrate that our concrete framework can be used at early stages of game design and development by instantiating it over an in-house game to aid the game balancing process. Instead of only relying on beta testers feedback, we collect and analyze the cause-and-effect relationships emerged during game sessions played by twelve invited subjects. The analysis is used to answer a set of questions game designers normally ask [18][4] (e.g., "How does the level of challenge increase as the player succeeds?").

This paper extends a prior conference paper [19] by demonstrating how provenance can be used in early stages of game development to aid the game design in the process of balancing a game. This analysis is mapped to questions game designs normally ask during the balancing process. The steps discussed here could be used with minor adjustments for other games.

This paper is organized as follows. The Section 2 presents related work and Section 3 provides background information as well as our proposed PinG framework. Section 4 presents the PinG framework usage and analysis over two existing games. Section 5 discuss the execution of the experiment, subjects' characteristics, and the design of the developed game. The results and discussion of the data game provenance analysis are also presented in the same section. Finally, Section 6 concludes this work, pointing out future works.

2. Related work

The literature adopts different terms for **tracked game data**, such as gameplay data, logged data, play traces, and telemetry data. Moreover, the process of analyzing such data, referenced here as **game analytics**, is also named in different ways, such as gameplay visualization, visual data mining, and game session analysis. In this section, we kept the original terms of each work, as they are usually reflected in the approaches' names.

Joslin [10] proposed the *Gameplay Visualization Manifesto* (GVM), which is a framework for gameplay data logging that uncovers gameplay events by attaching logging methods in the game objects responsible for generating relevant events during the game. The event model is the basis for the game data logging framework. It encapsulates the information that is desired by users and classifies the events into three groups: immersion, quest, and social. The immersion group represents events related to increasing the player's sensation of being involved in the game flux. The quest group represents events related to quest creation, execution, and analysis. Lastly, the social group represents events related to social factors in the game, such as group meeting or interaction with other characters.

The main application of GVM is for collecting game metrics, such as player deaths, position, time spent in available features (*e.g.*, crafting and fighting), item usage (*e.g.*, equipment), actions performed, and player enjoyment. Therefore, GVM does not track cause-and-effect relationships. It tracks only the executed actions along with their timestamp and location, in addition to character attributes and equipment.

Kim et al. [11] proposed the *Tracking Real-Time User Experience* (TRUE) approach that combines human–computer interaction (HCI) instrumentation, which collects *user initiated events* (UIEs), and log file analysis techniques in order to automatically record user interactions with games. Thus, TRUE can capture behavioral data and the attitudinal information behind the decisions made by the player in order to obtain better understanding of the context of each captured behavior.

Nevertheless, the designer still needs to infer the reasons behind the elements that led to an outcome. This occurs because the contextual information is only extra attributes that were tracked during the execution of the action and not actual relationships between events. Thus it does not capture cause-and-effect relationships. The cause-and-effect relationships must be inferred by the designer when analyzing the logged data. Moreover, TRUE was designed for the industry and is not easily available for indie companies. Even though we did not explore attitudinal data with PinG, it can be trivially incorporated in our approach as attributes for the player's actions or by creating specific activity vertices only for the attitudinal data when they are captured.

Playtracer [8], which is a visual tool designed to illustrate how groups of players move through the game space, aids the designer by tracking game states and showing common pathways and alternatives that players used to succeed or fail in their tasks, identifying pitfalls and anomalies in the scene. Nonetheless, *Playtracer* does not consider temporal information and does not preserve the order of the states visited by players when he/she revisits the same state. Moreover, incorporating *Playtracer* in the game design is challenging because it requires designers to define a state distance metric and identify relevant states.

Play-Graph [7] captures and illustrates the sequence of states and the actions that caused the player's state changes over the course of the game. In the Play-Graph context, a game state describes a certain configuration of the game or an entity, while actions consist on player interactions within the game, such as shooting, jumping, or using an object. In this concept, a game is viewed as a finite state machine with a finite number of states and transitions between them. The states are composed of a set of attributes from the game and players trigger actions at some specific points in the game. However, due to the nature of how the data is structured in Play-Graph, the understanding of player behavior is guided by the player progression in the game (*e.g.*, killed a boss), and not by how he/she interacted with the world (*e.g.*, combat rounds from the battle against the boss). From the available documentation, there is no way to determine interactions or influences. Only the changes from one state to another, caused by an action executed by the player, can be identified. Conversely, influences in the player's action, such as an influence from another character that affected the transition of one state to another, are not present in the graph (there are no edges linking edges).

According to Fernandez-Vara [20], different analyses need to be performed in order to increase the video game quality. Such analyses include understanding the game balancing in order to better attune for the vast majority of players. This process can be facilitated by using the collected game data. Furthermore, balancing also impacts how the player perceives the game difficulty. The analysis presented by Fernandez-Vara can be performed automatically to adapt the game difficulty to match the current player's skills. According to Black and Hickey [21], player's profile can change progressively or suffer immediate changes. The former is referred as evolutionary adaptation while the latter is referred as revolutionary adaptation. There are a plenty of approaches designed for performing dynamic difficulty adjustments

[4] In the book, Lens #31 (The Lens of Challenge), Schell defines a set of questions advised to be used while performing game balancing.

[3], ranging from the simplest to more complex ones. While the simplest can consider just state variables for performing such adjustments [22], more complex approaches for dynamic difficulty adjustments include using the player emotional state [23], procedural level design balancing [24], player modelling [21], and artificial intelligence [25,26] among others. The main drawback of them consists on the need of processing data to infer cause-and-effect relationships during analysis.

3. PinG: Provenance in games

The *Provenance in Games* (PinG) conceptual framework [12] was developed to map provenance concepts to the context of games. PinG was based on the PROV model [27], which provides the basis for specifying information that was involved in creating or influencing a particular object. Thus, PinG provides a mapping of elements from the provenance domain to the corresponding elements in a game domain, relating each data type of the provenance graph to typical elements found in games. In the game context, the provenance graph shows actions performed by characters (player or non-player) and events that occurred during game sessions and the causal dependencies among these actions or events. It is important to notice that the edges' orientation in the provenance graph goes from the present to the past, instead of the common orientation used in graphs, which are from the past to the future.

In the context of provenance, *entities* are defined as physical or digital objects. In the PinG approach, they are mapped into game objects without autonomous behavior. In provenance, an *agent* corresponds to a person, an organization, or anything with responsibilities. In the game context, agents are mapped into characters present in the game or game objects with autonomous behavior, such as event controllers, plot triggers, or the game's artificial intelligence overseer that manages the plot. Therefore, *agents* represent elements capable of making decisions or that have responsibilities in the game, while *entities* represent objects with no autonomous behavior. Lastly, *activities* are defined as actions taken by *agents* or interactions with *entities*. In the game context, *activities* are defined as actions executed or events that occurred throughout the game, such as attacking, dodging, and jumping.

The information collected during the game is used for the generation of the provenance graph, which is, in turn, used by a visualization tool. In other words, the information collected throughout the game session is the information displayed by the provenance graph for analysis. Thus, all relevant data should be registered, preferentially at a fine grain. The way of measuring relevance varies from game to game, but ideally, it is any information deemed relevant by the game designer that can be used to aid the analysis process.

However, Kohwalter et al. [16] implemented the provenance data gathering directly in the game as a prototype. In the work described by this paper, we created an independent and generic framework for Unity that is capable of gathering provenance during a game session. Thus, our provenance gathering framework is a domain-independent and low-coupling solution. Our framework is for the game engine Unity, which is written in *UnityScript* (a version of JavaScript used by Unity) that provides easier provenance extraction, requiring minimal coding in the game's existing components. The proposed framework has three different types of modules: seven *Core* modules, one *Interface* module, and six *Auxiliary* modules.

Fig. 1 illustrates a simplified class diagram for this framework, which we named as PinGU (PinG for Unity). *Core* classes are in yellow, *Interface* classes are in light blue, and *Auxiliary* classes are in orange. The *Core* classes represent the original infrastructure of PinG and are responsible for provenance information management, making everything transparent to the game designer. Analogously, it can be referenced as the provenance "server". Behind the scenes, the *Provenance Controller* class manages the creation of new vertices and edges and links them in the provenance graph. Meanwhile, the *Influence Controller* class manages the cause-and-effect relationships (influence edges), dealing with possible influences and passing it to the *Provenance Controller* class when they actually materialize in the game. The *Provenance Container* class exports the data to a XML file.

The *Interface* classes are the gateway between the game and the *Core* classes. While the *Core* classes can be seen as the server, the *Interface* classes can be seen as the client application. The *Provenance Extractor* class is where all provenance-gathering operations must pass through in order to reach the provenance-managing unit (or server). The *Auxiliary* classes contain pre-defined functions customized for a specific behavior, making easier to implement the provenance gathering.

3.1. Integrating PinGU into an existing game

A game developer can use PinGU to capture provenance data from a game by following the four stages described in this section: (1) adding the provenance controllers in the scene, (2) attaching the provenance extractors in each agent, (3) analyzing the game design document to extract knowledge for the provenance tracking, and (4) creating and attaching the provenance tracking functions. We use the game *2D Platformer Tutorial*[5] from Unity as a running example of the PinGU integration. Fig. 2 shows a screenshot of the game where the player has to kill aliens to gain score points. The game has two different types of enemies and the player can collect two different types of items to aid in his fight (health and ammunition items).

The **first stage** of usage consists of creating a game object in the scene to act as a centralizing server for the provenance information. This game object will have two attached classes: *ProvenanceController* and *InfluenceController*, which is illustrated in Fig. 3(a). As said earlier, both classes are used to manage all provenance information and graph generation, thus only one instance of each are necessary for each game scene. If the game is comprised of multiple scenes, then each scene will have its own provenance graph. These two classes use the other *Core* classes, which act as libraries and must not to be included in the scene.

The **second stage** is to attach the ProvenanceExtractor class in each character or entity in the game (i.e. NPCs, player, interactive objects, prefabs) and link it to the object created in the first step. This class is responsible for creating all the provenance vertices for the game entity that is attached to and then passing these vertices to the *ProvenanceController* to insert it in the graph. Fig. 3(b) illustrates an example of adding the class to the *Hero* game object, which is the player's avatar from the 2D Platformer.

The **third stage** is to identify the actions and their interactions with other actions in the game design document. In the running example, we identify the existing classes that contain the actions that we want to track, which are illustrated in Fig. 4. The same figure also shows a summary of each selected class and their responsibility in the game, grouped by the identified agents (*i.e., Enemy, PlayerControl, PickupSpawner*). The classes for the agents also contain additional actions, such as spawning item and movement.

The **fourth stage** is creating the domain-specific *provenance tracking functions* and attaching it to each entity in the game that has the *ProvenanceExtractor* module. Each existing module should have a *provenance function* for each possible action that the entity can perform and that we are interested in tracking.

Unfortunately, it is necessary to create these provenance function calls due to domain contextual information. However, all these *provenance functions* are small and simple, following the same four-step recipe and changing only the context information used during each step:

1. Add game-related attributes (*e.g.*, health points, experience points, etc.);
2. Create the appropriate vertex (Activity, Agent, or Entity);

[5] https://www.assetstore.unity3d.com/en/#!/content/11228.

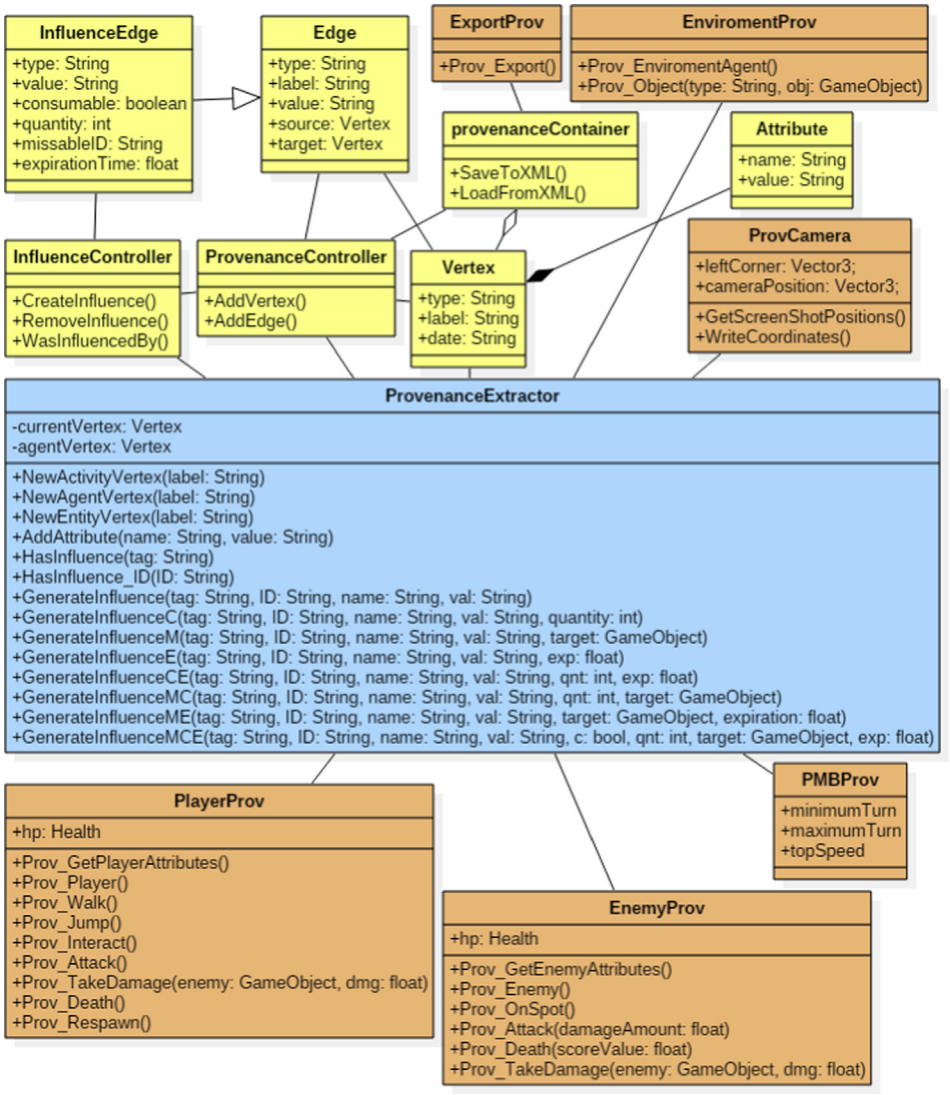


Fig. 1. Simplified class diagram for PinGU.

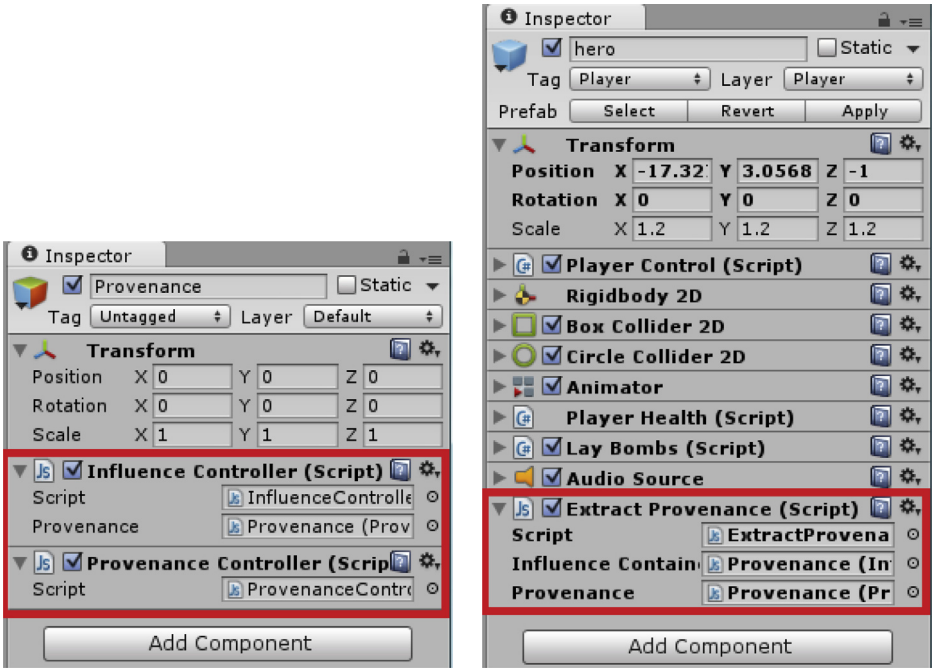
Fig. 2. 2D Platformer game. Source: https://www.assetstore.unity3d.com/en/#!/content/11228.

3. Check for influences (if applicable);
4. Generate influence (if applicable).

The first step is used to configure the desired information to be extracted during the execution of each action or event. They will appear at the graph's vertices as attributes. Unity already provides default attributes, such as location, tag, object name. However, game-sensitive attributes such as health points, magic points, and player score must be manually added by the *AddAttribute*(< name > , < value >) function of *ProvenanceExtractor* class. After adding the desired attributes, the second step creates the provenance vertex and places it in the graph. This vertex can be any of the three provenance types and must be specified by the user by calling the *NewActivityVertex*, *NewAgentVertex*, or *NewEntityVertex* functions.

The third and fourth steps are related to influence. The third step is used to verify if there is any influence that can affect the current action. If so, they are automatically inserted in the graph as an edge connecting the respective vertices. This verification can be made by a tag (*HasInfluence*(< tag >)), which is used to group a collection of influences that has something in common, or by an influence ID (*HasInfluence_ID*(< ID >)).

The forth step is responsible for creating influences (*GenerateInfluence*), so they can be used by the third step. Influences can be created with some restrictions: They can expire when a certain time passes (*e.g.*, spell duration), leading to the E (expire) suffix at the function (*i.e., GenerateInfluenceE*), or after a number of times used (*e.g.*, spell that block the next attacks) leading to the C (consumable) suffix (*i.e., GenerateInfluenceC*), or both (*GenerateInfluenceCE*). There is another type of influence that can be combined with the restrictions above, which represents something that was expected to happen but for some reason, it did not. For example, there is a health item in the scene that the player is supposed to get, but he forgot or skipped it. Thus, if the player did not get it, then an influence is generated saying that the



(a) 1st stage for PinG integration, showing the *Provenance* game object and its scripts.

(b) 2nd stage for PinG integration, showing the insertion of the provenance tracking class in existing agents and entities.

Fig. 3. PinG integration.

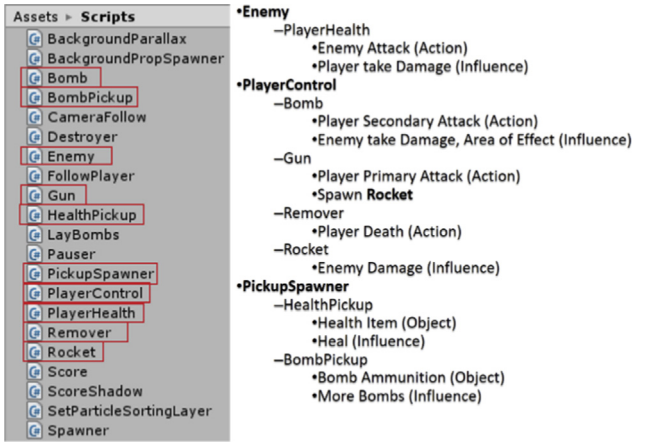


Fig. 4. 3rd stage for PinG integration, showing the 2D Platformer classes and Game Design.

player "missed" the item. However, if the player did, in fact, get the item, then the normal influence (effect of getting the item) occurs. For those, the function has the suffix M ("missable") (*i.e., GenerateInfluenceMC, GenerateInfluenceMCE*).

Listing 1 shows an example of a provenance function for our running example of one of the possible actions that can be executed by an enemy. The calls used in the *Prov_Attack* are implemented in the *ProvenanceExtractor* (*NewActivityVertex, HasInfluence, GenerateInfluenceCE*), with the exception of *Prov_GetEnemyAttributes*, which is domain related and the developer need to specify the desired attributes for tracking, besides the default attributes from Unity (*i.e.*, Tag, object name, object coordinates). This is accomplished by creating a function (e.g., *Prov_GetEnemyAttributes* from the auxiliary classes) that invokes the function *AddAttribute* from *ProvenanceExtractor* by passing the attribute name and value for each attribute, as illustrated by Listing 2.

After creating the necessary *provenance functions* for their respective game objects, the next step is to incorporate the function calls in existing game classes in order to register the provenance information. All this process becomes trivial if the developers have a detailed game design document stating all the possible actions that can be executed in the game along with their purpose. The action list shows the actions that are desired to be tracked and the necessary provenance functions that need to be made. Meanwhile, the action's purpose gives us insights on the influences that they can generate during or after executing the action.

Listing 3 shows an example of code insertion in an existing game module responsible for controlling the artificial intelligence (AI) of enemy characters in the game. The "damageAmount" is a configurable variable from the original class that states the damage the attack will cause. We inserted the provenance call for the *Prov_Attack* function, whose code appears in Listing 1, in the function responsible to make the enemy AI fire at the player. We added a package of auxiliary classes that, depending on the type of the game, does the majority of the work and requires only coding the function call in the existing game classes. Furthermore, they can also be used as a guiding example in cases that the desired action is not already implemented. These classes are *PBMProv, PlayerProv, EnemyProv,* and *EnviromentProv*, and each is customized for the particular type they represent (Car-related movements, Player, Enemy, and Environment).

The last step is to add a provenance export function to an event so it can save the current provenance graph to an external XML file when the designated event is executed (*e.g.*, player's death, completing the level). Listing 4 illustrates the *provenance functions* for our running example responsible for exporting the tracked data, which is linked to the player's death, and Listing 5 shows the insertion of the *provenance function* call to track the information.

Fig. 5 shows an example of the generated provenance graph from the tracked actions executed during a game session, which was

```
public string Prov_Attack(float damageAmount)
{
        Prov_GetEnemyAttributes();
        prov.NewActivityVertex("Attacking","");
        prov.GenerateInfluenceC("Player",
            ↪ this.GetInstanceID().ToString(), "Damage",
            ↪ (-damageAmount).ToString());
        return this.GetInstanceID().ToString();
}
```

Listing 1. PinG code for tracking game data. Orange text in the code is domain-related.

```
public void Prov_GetEnemyAttributes()
{
        prov.AddAttribute("Health", HP.ToString());
}
```

Listing 2. Example of a provenance function for tracking attributes.

rendered using *Prov Viewer*. We can see in this graph the player's and each enemies' actions and how they interacted with each other by looking at the vertical colored edges.

3.2. Capturing game scene

We also implemented a specialized camera module in order to simplify the process of capturing the game map to use it in combination with the provenance graph. This camera is orthographic, which preserves the dimensions and does not change coordinates to accommodate the perspective of the viewer. Thus, this camera needs to be placed either directly above the game scene or laterally (for platform games), allowing it to capture the entire map. This module automatically captures the screenshot of the scene and the necessary data required to align the provenance graph, which uses world space coordinates, with the captured map, which uses pixel position. The screenshot resolution can also be adjusted in the module.

The camera module captures the camera's world position (*cameraPosition*) and the camera's upper left corner coordinates in world position (*leftCorner*). The camera's position is used to translate the game map in order to align it with the graph and is easily obtained by getting the position of the camera in world space. The second information is used to scale the graph to match the picture and is captured by converting the camera position from viewport space to world space, which is the upper left corner.

In order to align the graph with the map, it is necessary to find a scale factor, that can be trivially be calculated by Eq. (1). The equation uses half the screenshot's picture width to determine the distance between the center, which is the position in the picture where the camera is when the screenshot was taken, to the left edge to properly scale the graph.

$$scaleFactor = \frac{0.5 \times pictureWidth}{leftCorner_x - cameraPosition_x} \quad (1)$$

The *scaleFactor* is used to transform the world coordinates captured from the provenance data to pixel coordinates used in the screenshot of

```
function Fire () {
        if (weaponBehaviours[nextWeaponToFire]) {
340             weaponBehaviours[nextWeaponToFire].SendMessage ("Fire");
                nextWeaponToFire = (nextWeaponToFire + 1) %
                    ↪ weaponBehaviours.Length;
                lastFireTime = Time.time;

345             // Provenance
                prov.Prov_Attack(damageAmount);
        }
}
```

Listing 3. Provenance function call insertion into existing classes.

```
public void Prov_Death()
{
        Score score = GameObject.Find("Score").GetComponent<Score>();

        prov.AddAttribute("Health", "0");
        prov.AddAttribute("Score", score.score.ToString());
        prov.NewActivityVertex("Death","Drowned");
        Prov_Export();
}

void Prov_Export()
{
        Debug.Log ("Exported");
        GameObject ProvObj = GameObject.Find("Provenance");
        ProvenanceController prov =
            ↪ ProvObj.GetComponent<ProvenanceController>();
        prov.Save("2D_Provenance");
}
```

Listing 4. Provenance function for the player's death action.

the game map. Therefore, the game designer only needs to position the orthographic camera in the game scene and add the camera module in order to capture the entire map and the necessary data. After that, the designer can use the coordinates captured by the module and the screenshot in a visualization tool.

3.3. Provenance graph visualization

One of the purposes of collecting provenance data is to be able to generate a provenance graph to aid the developer in analyzing and inferring the reasons for the outcomes. After incorporating the PinGU approach into an existing game, the provenance data is captured and stored while a game session is being played. Afterwards, users can generate a provenance graph for that specific game session.

The generated provenance graph is exported to a simple XML file containing a list of vertices and edges in the graph. This data can be used for data mining, exploration, and visualization. For this work, we employ the open-source provenance visualization tool named *Prov Viewer*[6] [17], which uses a graph framework to allow detailed rendering and visual data analysis and exploration of the provenance information. The tool provides many visualization and manipulation features: (1) collapsing, highlighting the relevant information in the graph; (2) filtering, removing information that is not relevant for a given analysis; (3) graph merge, integrating the analysis of multiple game sessions; (4) specialized layouts, organizing the graph in a more understandable way; (5) domain configuration, customizing the visualization for specific needs; and (6) shapes, sizes, and colors, supporting a clear distinction between information types. Fig. 6 illustrates the tool's architecture, highlighting its main features.

When evaluating tracked attributes, *Prov Viewer* uses traffic light scheme to quickly differentiate values, thus changing vertex color to the appropriate shade. The shades vary from red to green, with yellow as the middle term. Similarly, edges also use shades to distinguish values of the same type (*e.g.*, damage), as well as thickness to show how strong the relationship is. Bright red represents negative values, bright green represents positive values, and darker shades represent values near zero. This feature allows the user to quickly identify strong influences in the graph just by looking at the edge's thickness and their color. Fig. 7 illustrates some of these visualizations features in action.

The tool also has a spatial layout that organizes the vertices in the graph by their spatial coordinates and can be used for spatial or geo-referencing the data. The layout supports the usage of an orthographic image, which is captured in the PinGU framework. This is particularly useful for corresponding elements with other graphical representations, such as a map of the game scene. When using the spatial layout in conjunction with a background image, the user can see where each tracked event occurred just by looking at the graph's placement in the image. All the graph images in the following sections were rendered using *Prov Viewer*.

4. Case studies on the instantiation of PinGU over existing games

The following sub-sections present two open-source game samples (*Car Tutorial*[7] and *Angry Bots*[8]) where we demonstrate the generated provenance graphs by incorporating PinGU in existing games. In the first game, we focus on showing that the provenance data can facilitate the graph analysis on how previous actions or events affect future actions. We also show how the provenance graph evolves when the game has multiple cycles. In the second game, we show another case of

[6] http://gems-uff.github.io/prov-viewer/.

[7] https://www.assetstore.unity3d.com/en/#!/content/10.

[8] https://www.assetstore.unity3d.com/en/#!/content/12175.

```
void OnTriggerEnter2D(Collider2D col)
{
        // If the player hits the trigger...
        if(col.gameObject.tag == "Provenance")
        {
                // .. stop the camera tracking the player
                GameObject.FindGameObjectWithTag("MainCamera").
                    GetComponent<CameraFollow>().enabled = false;

                // .. stop the Health Bar following the player
                if(GameObject.FindGameObjectWithTag("HealthBar").activeSelf)
                {
                        GameObject.FindGameObjectWithTag("HealthBar").
                            SetActive(false);
                }

                // ... instantiate the splash where the player falls in.
                Instantiate(splash, col.transform.position,
                    ↪ transform.rotation);
                // ... destroy the player.
                Destroy (col.gameObject);

                 //Provenance Death
                Prov_Death(col.gameObject);

                // ... reload the level.
                StartCoroutine("ReloadGame");
        }
        else
        {
                // ... instantiate the splash where the enemy falls in.
                Instantiate(splash, col.transform.position,
                    ↪ transform.rotation);

                // Destroy the enemy.
                Destroy (col.gameObject);
        }
}
```

Listing 5. Fragment of the original *Remover* module: Added the *provenance function* call in the player's death.

provenance data with a different genre of game, allowing for easy identification of sections that were not explored by the player and where he/she had more difficulty. We did not modify the games in any way nor added new features besides coupling with the PinGU, which is only responsible for tracking provenance data. Both case studies use *Prov Viewer* tool for visualizing the provenance graphs.

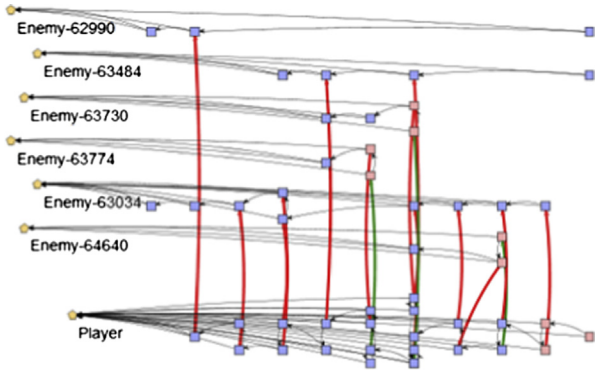


Fig. 5. Example of the generated graph for the *2D Platformer*.

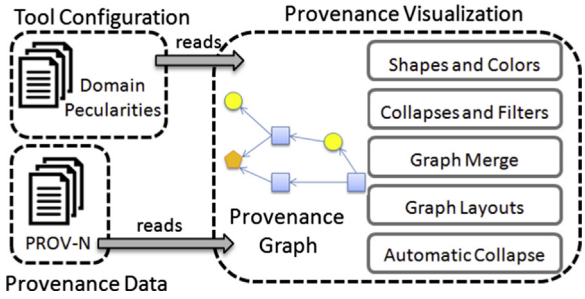


Fig. 6. *Prov Viewer's* high-level architecture (from [17]).

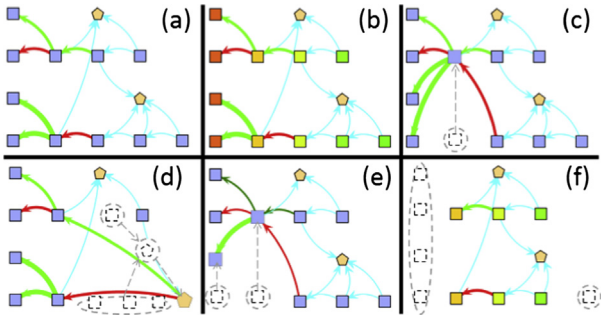


Fig. 7. (**a**) Original graph; (**b**) graph with a color schema; (**c**) collapse of two activities; (**d**) collapsing of the agent's activities; (**e**) graph *c* after another collapse; and (**f**) temporal filter (from [17]).

4.1. Car tutorial

The first case study is the Car Tutorial, freely available at the Unity asset store. This tutorial has only one racetrack and focuses on the arcade style racing game. In addition, there is no implemented AI for opponent cars. Fig. 8 shows a screenshot of the game. Following the conceptual framework, PinG tracks events and actions executed during the game session, along with their effects on other events, to compose the provenance graph (*e.g.*, crashing the car, pressing the car's brake).

Fig. 8. Car Tutorial screenshot. Source: https://www.assetstore.unity3d.com/en/#!/content/10.

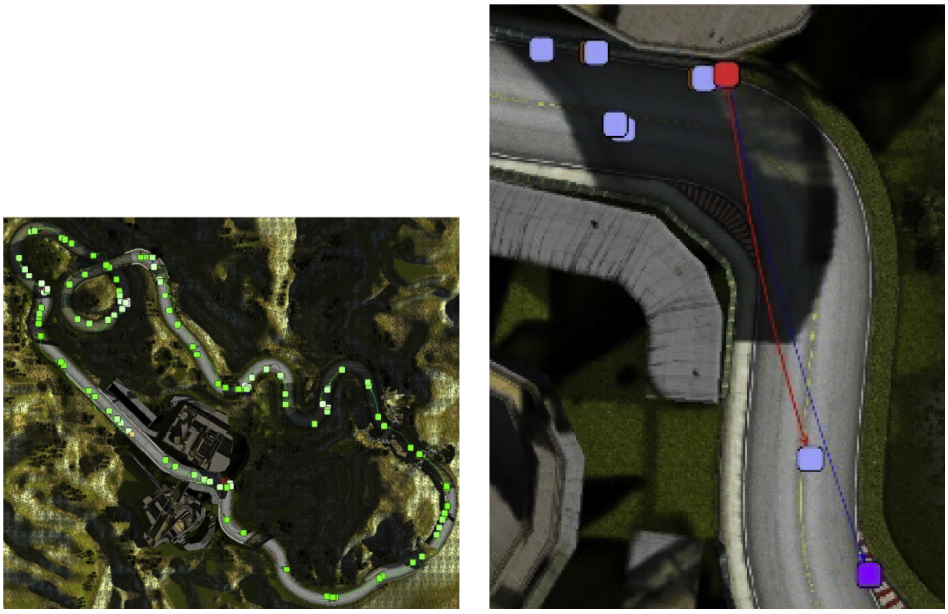
We can use the car's coordinates on the track to plot the graph so that it is possible to visualize where the player was when the action was executed. This visualization also allows the designer to quickly identify which sections of the track the player had trouble. Thus, we can take advantage of spatial-referencing the data during the provenance visualization. We used a screenshot of the game map taken by our camera module with dimensions of 1070 × 802.

Fig. 9(a) shows the provenance graph of one game session, using the car's coordinates and the track's picture as background. This graph is composed of 169 vertices and 867 edges extracted from a 107-s game session, which represents one complete lap in the track. The vertices are colored according to the car's speed (gradient from white when close to zero and green for high values) and the visible edges are the speed delta between vertices.

We can quickly identify sections of the track that the player may have had issues, either by reducing the speed too much or by crashing, by just looking at the plotted graph in the race track. As an example, Fig. 9(b) shows a zoomed section of the graph to better illustrate the reasons behind a car crash. The zoomed section of the graph has a different vertex-coloring scheme to differentiate events. By analyzing it, we can see that the car crash (red vertex) was influenced by two factors. The first one was on the previous curve, where the car lost contact with the ground (purple vertex with a blue edge linking the crash) after passing through the rumble strips at the end of the maneuver, thus preventing the player to prepare for the following turn. The second reason was that the player was too fast, as indicated by the red edge from the blue vertex, which is a reduction of the car's turn rate due to high speed.

Using the tracked telemetry data from other laps of the race, we can begin to detect patterns during the game session or even compare the player's performance between laps. This analysis can also be extended to different game sessions by comparing the generated provenance graphs. Fig. 10(a) illustrates an example of the generated provenance graph when gathering data from multiple laps during a single play session, enabling the designer to detect behavioral patterns and locations where the players are struggling the most. For example, Fig. 10(b) shows a section of the track that is characterized by having multiple curves in the track. We can see the player's performance during each lap of the race, where each lap is represented by a different edge color. The first, second, and third laps are presented by red, green, and blue edges respectively. Moreover, the first and last vertices of each lap are marked with circles of the same color as the edge and the timestamps are represented by the yellow numbers together with the vertex. As we can see, the player had approximately the same speed in all laps due to having the same shade of green when entering this section of the track. However, the player took fifteen seconds to pass through this section of the track on his first lap (52–37), seventeen seconds during the second lap (131–114), and ten seconds on the third lap (200–190).

By analyzing Fig. 10(b), we can see a purple edge that represented the reason behind the crash in the first lap (marked by the purple circle). This purple edge represents a cause-and-effect relationship, showing that the crash happened because the player passed through rumble strips (brown circle) and, as a result, lost car stability and could not complete the turn. Furthermore, notice the steep angles the player had to make due to his positioning in each curve. During the second lap (green edges), the player tried to avoid the crash by reducing speed. However, the player reduced too much speed to enter the second curve (white-green vertices). During the third lap (blue edges), the player managed to improve his performance. He/she avoided any crashes by better positioning the car before each curve, reducing the necessary angle to make the turn while maintaining a nearly constant speed.



(a) Spatial referencing provenance data. (b) Influences behind a car crash.

Fig. 9. Car Tutorial provenance graph.

4.2. Angry bots

We conducted a second case study using a very different style of game, called Angry Bots, also freely available at the Unity asset store. A screenshot of the game is presented in Fig. 11. Angry Bots belongs to the hack-and-slash genre, being a top-down action shooter. In the available scenario, the player has to face enemy robots and interact with the environment in order to complete the level.

Fig. 12 illustrates one of the possible visualizations of the provenance data gathered by our framework, showing the vertex visualization scheme for the player's health attribute value (vertex color using a traffic light scheme) and the edges that influence in it (green and red edges) as the game progresses. Blue vertices represent other characters in the game (enemies), blue edges represent the chronological order of events, and green edges represent player's health generation due to his passive regeneration ability. By analyzing Fig. 12, we can see the chronology of events, regions visited by the player, sections where more action happened, places where the player engaged in battle, and when

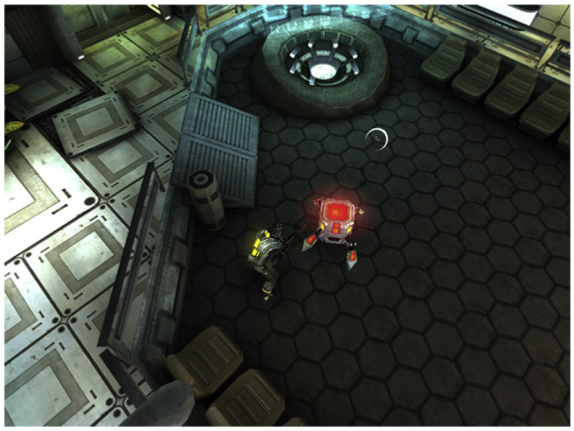


Fig. 11. Angry Bots screenshot. Source: https://www.assetstore.unity3d.com/en/#!/content/12175.

(a) Provenance graph from multiple laps. (b) Zoomed section from yellow rectangle on Figure 10(a).

Fig. 10. Car Tutorial provenance graph from multiple laps.

Fig. 12. Picture of the entire graph. Vertex coloring based on player's Health attribute.

the player suffered heavy health loss. In this game, we used a screenshot taken by our camera with the dimensions of 4280 × 3208 in order to show that the figure size does not affect the graph alignment process. This increase of resolution allows for a higher detail of the game scene visualization when zooming the graph during analysis.

Considering that the player recovers health periodically, it is possible to infer that the cause of some deaths was the rush through the level without waiting to recover health or because of a tough enemy. Fig. 13 illustrates the first case, where the player tried to rush through the game without waiting to regenerate the player's health, which was lost from the previous battles. The light blue arrows were added to the figure to highlight the player's general movement and does not belong to the provenance data.

After the player engaged an enemy in a major battle, which the player didn't leave unscathed by looking at the orange vertices, the

Fig. 13. Player's health when trying to rush the game.

player continued advancing through the level. Then, on the player's third major engagement, where he/she was still wounded by looking at the orange vertices, the player lost the majority of his remaining health, as illustrated by the following red vertices. Even though the player was low on health, he managed to dispatch his enemies on the forth battle without losing a single health point (no red edges). However, the player continued pressing on without resting, which would allow for him to gradually restore his lost health points before his next engagement, until dying on the next battle when the player got hit by the enemy (Battle #5).

Fig. 14(a), (b), and (c) illustrate the second case, showing the sequence of events that led the player to a tough engagement (Fig. 14(c)). By analyzing the picture, we can see that the player started these events (Fig. 14(a)) with good health (green vertices), leaving the first battle slightly injured (yellow vertex). A few moments later he encountered another enemy in a side room (Fig. 14(b)), where once again he overcame the enemy with only minor wounds (the vertex is still yellow). However, just when he left the room, the player was ambushed by another enemy that was patrolling the corridor (the new blue vertices in the corridor from Fig. 14(c)). This enemy was a mech, which is much tougher than a regular enemy (notice the high number of dark red edges that represent player doing damage to the enemy). This battle resulted in the player's death after getting hit by two rockets (Fig. 14(d)) followed by his resurrection shortly after (green vertex in the bottom of Fig. 14(c) that is linking the green edge to a red vertex).

Fig. 15 illustrates the moments when the player died, which are marked by red circles. Meanwhile, the orange circles illustrate the player "refreshed" state after resurrecting, as well as the resurrected location. Both situations have a green edge linking the player's death to the resurrection, which shows that his health went from zero (red vertex) to maximum (green vertex) after resurrecting. Notice that the player actually died three times trying to beat the mech enemy from Fig. 14(c) before finally defeating it.

5. A case study on the use of provenance during the design of a game from scratch

In this section, we discuss the usage of provenance in the early stages of game development using PinGU when implementing the *MorphWing* game from scratch. Although in this paper we use the analysis performed for helping the game balancing process, we believe that designing a game with provenance may also be useful for other purposes such as dynamic level of difficulty and bug identification.

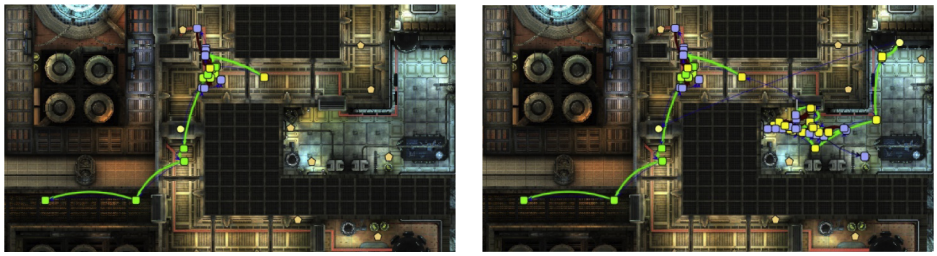
The main objective of this section is to evaluate how useful is the usage of provenance for answering game design questions in early stages of game development. We selected four design questions (DQ) frequently used by game designers for balancing a game for this experiment. These four questions have been answered through provenance data collected from 12 participants. The four design questions used to guide this analysis are detailed below:

DQ1: How the player expertise increases with time? *This DQ assesses how the player masters the game after playing it for a certain number of times. The answer to this question can be used to tailor how challenges should increase along the time.*

DQ2: How is the impact of the enemies' behavior on the player performance? *This DQ analyzes the influence of different types of enemies on the players. This analysis can suggest how and when each type of enemy should be placed in the game.*

DQ3: How the collectible items impact positively or negatively on the player experience? *This DQ verifies the impact of both positive and negative spawned items that are found in the game, as well as how they influence the players' performance. As in the DQ2, the answer to this question can tailor how and when the collectible items should be spawned in the game.*

DQ4: How the participants perceive the difficulty of the game as a



(a) Sequence of events of the player exploring a section of the map and confronting an enemy.

(b) Continuation of the sequence of events from Figure 14(a) with a second engagement inside a room.

(c) Continuation of the events from Figure 14(b) that led the player to a tough confrontation that resulted in his death.

(d) A zoomed section from Figure 14(c) showing both the moments the player was hit by the enemy's rockets. Filtered to show only the edges that affected the player's Health.

Fig. 14. Sequence of events of the player from exploring a room to his death.

Fig. 15. Filtered graph showing the moments the Player died and was resurrected.

whole? *This DQ is aimed to check, through an interview, if the analyses and conclusions obtained from provenance are also perceived by the player.*

The following subsections describe *MorphWing*, the game implemented for demonstrating the early usage of provenance during development. Next the experiment design is explained, followed by results and discussions of the posed DQs.

5.1. MorphWing design

The *MorphWing* game was developed using the Unity game engine and our PinGU framework so participants could answer the design questions previously presented in this paper.*MorphWing* is a 2D game where the **main objective is to live as longer as possible**. It is important to stay clear with the game' main objective as all the following analysis will be performed with it in mind.

During the game, items are spawned throughout the game session, which can positively or negatively affect the player. A screenshot of the game is presented in Fig. 16.

In *MorphWing* the player starts with a total of ten health points and loses it while colliding with the enemy or being hit by enemy projectiles. In both cases, the player becomes invincible for a very short moment, nullifying all damage in this period, in order to give a chance for the player to recover. During the game, the player has two different type of weapons: a *Spread* and a *Heavy* weapon. The former produces

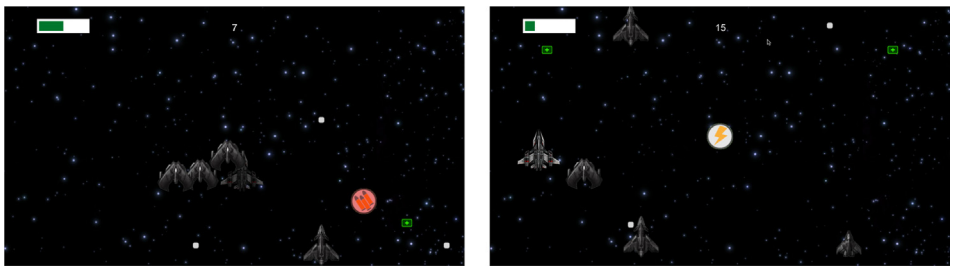


Fig. 16. The MorphWing Game.

three projectiles at once with different directions, each causing one hit point damage. The latter produces just one projectile, causing eight hit point damage. Besides their inflict damage difference, their availability for use is also different: *Spread* can be used five times in a second while *Heavy* just once per second.

MorphWing presents four different enemy types with a maximum of four enemies at the same time. This value was chosen as it presented a good challenge balancing in the initial development tests, without using provenance. The characteristics and behavior of each type of enemy are as follows:

- **Straight**: moves in a straight direction until it reaches the other side of the screen, disappearing afterwards. This enemy has a total of six health points and causes one hit point if collides with the player.
- **Chaser:** chases the player for colliding, causing two hit points of damage to the player. It has a total of ten health points.
- **Boomerang:** after appearing from a random corner of the screen, this enemy moves straight for a moment and then stops, shooting a bullet on the player's direction and moves back towards its spawn point, disappearing when reaching it. It starts with four health points, and both the *Boomerang's* bullet and its collision with the player inflict one damage hit point.
- **Round Shooter:** after appearing from a random corner of the screen, moves straight for a moment, stops, then shoots bullets clockwise in 8 directions on a circular pattern, starting from the top. It starts with five health points, and both the *Round Shooter's* bullet and its collision with the player inflict one damage hit point.

Items are spawned and remains visible for 4 s and are removed if not collected by the player. Except for the healing item, which is instantaneous, the effect from all other itens has duration of 2 s. In total, up to three items can be on the screen simultaneously. Each item has a different icon as well as color, with green or red tinted for positive and negative effects, respectively.

The following items are available in the game:

- **Healing:** recover two player's health points.
- **Control Reverser:** temporarily inverts the player's movement directions (up becomes down and vice versa, left becomes right and vice versa).
- **Damage Up:** temporarily increases the player's damage output by a factor of two.
- **Damage Down:** temporarily decreases the player's damage output by a factor of two.
- **Speed Up:** temporarily increases the player's movement speed by a factor of 1.5.
- **Speed Down:** temporarily decreases the player's movement speed by a factor of 1.5.

5.2. Experiment design

The experiment has been conducted with 12 participants (named P1 - P12), who never have played the game before. Fig. 17 presents their characteristics. All participants are aged between 16 and 25 years old.

The first step of the experiment involves giving initial instruction for each of the participants, explaining about the enemies in the game and their behavior as well as the available items and their effects, and answering questions that came out during this step. Following, the participants had the chance to play the game once in order to understand its mechanics. After this, the participants play three more sessions that were used for the analysis process. Finally, an exit interview was conducted with each participant, in order to answer DQ4. We fixed the enemy and items spawn position, as well as their number, across all sessions to minimize the bias from the randomized elements of the game. In this case, all three sessions have almost the same level of difficulty.

5.3. Results and discussion

We collected the generated provenance data from participants' sessions to analyze our proposed design questions, showing that collecting provenance in early design stages can be useful for balancing the game.

DQ1: How the player expertise increases with time?

In this design question, we aim to show how the player's expertise evolves through time. In order to answer this question, it becomes necessary to analyze how the player behaves along different sessions of the game by using some type of metric. Normally, a commonly used metric involves counting the total number of enemies hit by the player. Although this is an empirical observation, the game sessions of the test showed that it was a reasonable metric. Moreover, we defined a player's hit rate variable (*KH*), that varies across the time, using the formula $KH = \frac{K_E}{S_T}$, where K_E and S_T represent enemies hit and session time (in seconds), respectively. Fig. 18 shows how *KH* varies across the different sessions.

According to Fig. 18, it is possible to observe three important facts. The first one involves players getting better across the sessions. In this case, the provenance showed us that 7 out of 12 (58%) participants increased their *KH* from session 1 to session 3. Additionally, a second fact that can be observed thanks to the provenance is that the increment in skill among the participants presented a high variation. For instance, P1 and P3 differ their skill rate in about three hits/s in session 3. This indicates that players evolve at different rates. Finally, the provenance showed that some participants presented a decreasing *KH* along the sessions (P6, P9, P11, and P12). In the proposed game, it is essential for the player to move in the scenario for avoiding being hit by the enemies or collide with them. While looking in Fig. 19, which shows the provenance related to the total distance traveled for each participant, it is possible to notice that all these four participants decrease their distance traveled, being an easy target for the enemies.

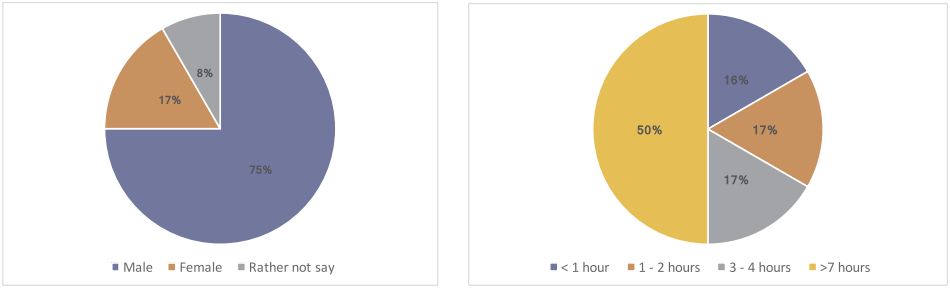
Using the provenance graph, we traced the survival time (T) and distance traveled (D) by the player, producing 12 tuples in the form <T,D> per session. For each session, we applied a correlation (using the Spearmans ρ) analysis for detecting a possible relationship between time and distance travelled. The result shows a positive correlation between them (0.79, 0.65, and 0.27 for session 1, 2, and 3, respectively) indicating that players who move frequently stay alive longer. Besides that, it is possible to observe a decreasing correlation from session 1 to session 3, indicating players survival time is getting less dependent to distance traveled. One possible reason for this behavior is players shooting getting more accurate, reducing the necessity to evade in order to avoid being hit by the enemies.

In order to demonstrate this fact, Fig. 20 shows the provenance graph of the longest (P12 - Session 3) and the shortest (P8 - Session 3) playing time. Orange pentagon outside the screen game field represents enemy's spawn points. According to this figure, it is possible to observe that P12 is scatted in almost all positions in the map (vertex colored in magenta), while P8 is concentrated in a small area of the map. This dynamic transition in the scenario allowed P12 to be more efficient in killing (represented by green vertices) and evading enemies than P8.

The presented analysis leads to a conclusion that the usage of provenance at early stages of game development can unveil how expertise changes among the players. At first, a strong correlation has been found between distance traveled and survival time, showing that movement is important to be successful in the game. However, this correlation decreases in future game sessions, indicating players found strategies other than keeping moving to be successful in the game.

DQ2: How is the impact of the enemies behavior on the player performance?

Each kind of enemy has different behavior and movements. In our experimental game this is present in the *Straight* enemy that moves in just one direction, and the *Chaser* enemy that pursuit the player until it



(a) Gender. (b) Playing time by week.

Fig. 17. Participant's characteristics.

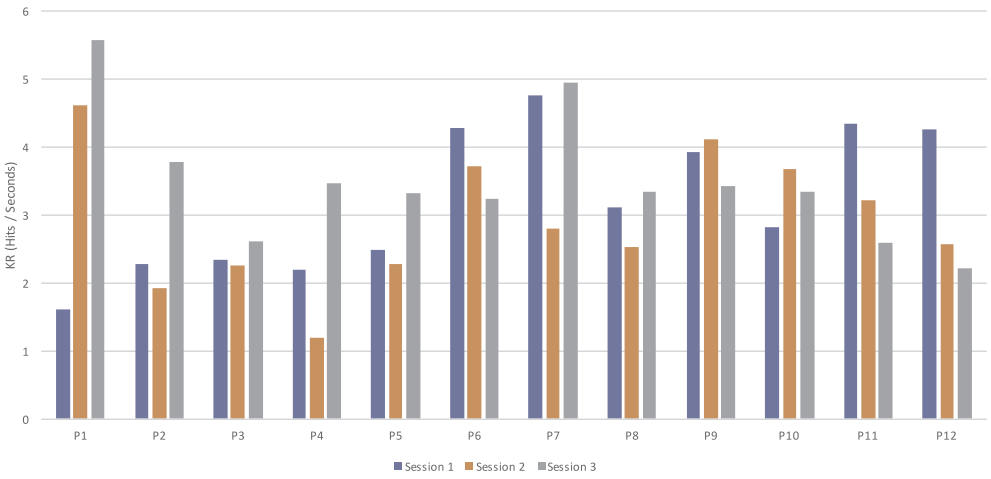


Fig. 18. Player's skill evolution across the sessions.

gets destroyed. Besides that, some enemies can shoot using different patterns. Most of the difficulty the player faces in the game is related to how strong each kind of enemy is as well as how many of them are on the field simultaneously. Due to this fact, the balance factor of the game considering the players abilities is closely related to the number of enemies on the screen and their strength.

In a first moment, it is important to understand how each type of enemy harms the player. One of the main characteristics of the developed game is having various instances of an enemy type on the field at the same time. In this case, such analysis involves understanding how these numbers influence the player's performance. We used the provenance graph for checking the number of times each type of enemy inflicted damage to the player for all the three sessions. This information can be seen in Fig. 21.

According to Fig. 21, it is possible to observe that *Chaser* enemy is the one that most inflicted damage to the player for the three sessions. Following, the *Boomerang* enemy is the second that most inflicted damage to the player.

Fig. 22 presents the interview conducted with each participant, asking about the most challenging enemy. The majority of the participants answered *Chaser* (83.3%), followed by *Round Shooter* (16.7%). In fact, according to Fig. 21, the *Chaser* is the one that most harmed the

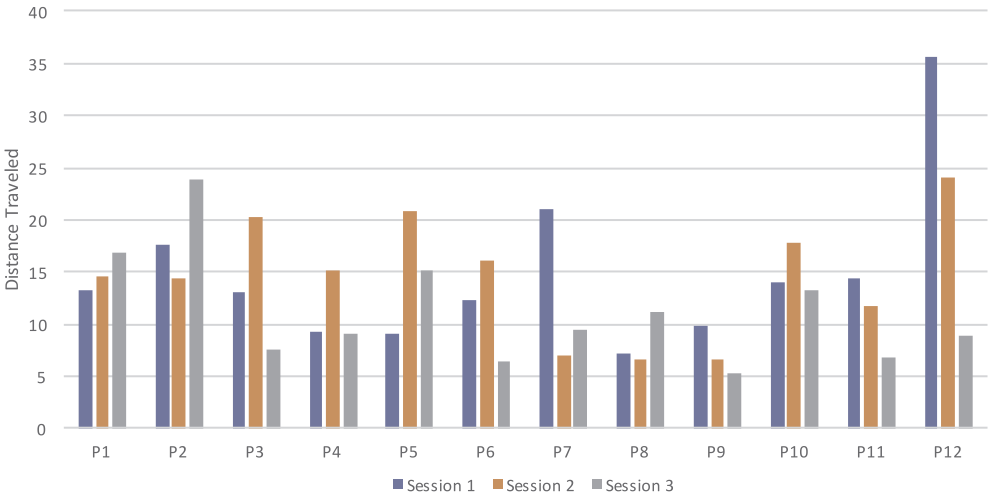
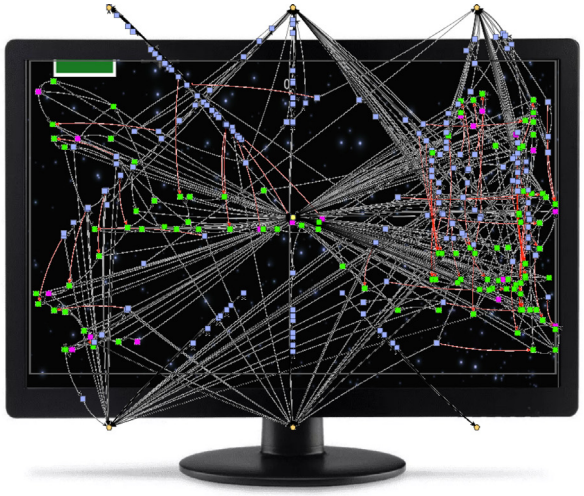
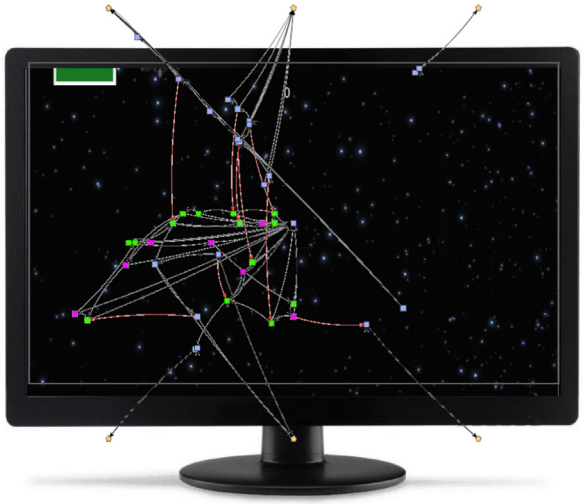


Fig. 19. Distance traveled by each participant in each session.



(a) Provenance of the participant with longest distance.



(b) Provenance of the participant with shortest distance.

Fig. 20. The longest and shorter distance travelled by participants.

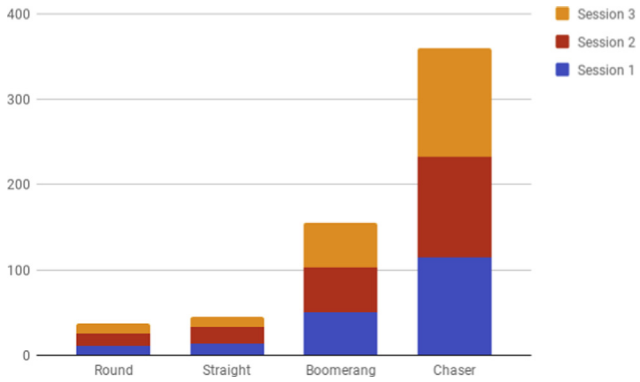


Fig. 21. Damage inflicted by each type of enemy for each session.

players in all sessions. This shows that the provenance analysis was able to correctly capture the participant's perception regarding the game difficulty.

It is important to state that different analysis can be performed using tracking counters inside the game. However, these counters must be conceived and instrumented *a priori* in the game's source code. In a situation where a new analysis must be performed but a counter is not being tracked, then the counter cannot be obtained. On the other hand,

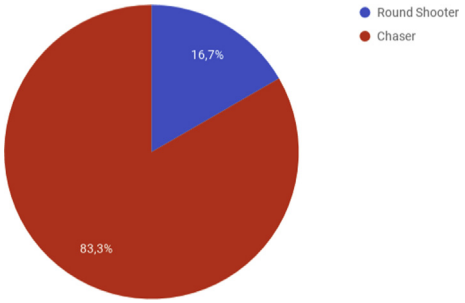


Fig. 22. The most challenging enemy according to the participants.

by using provenance, it becomes possible to get a new information even in the case where it has not been previously identified as required in the game design document.

DQ3: How the collectable items impact positively or negatively on the player experience?

One important factor in the player's performance is the items he/she collects. For instance, if the player is able to collect items that increase the damage caused to enemies, there are chances that the player lives longer and become able to collect more healing items. Thus, it becomes important to understand how these elements influence the player. Table 1 presents the number of spawned and collected items and the rate it represents in the game.

According to Table 1, it is possible to see that *Healing* item is the most spawned. However, only about half of them have been collected by the players. While observing the participants playing the game, it was possible to notice that most of them tried to collect the item but was not able due to their distance. One way to balance the game should be by spawning the item in a location next to the player or increase the time it stays on the screen depending on how much damage the player had taken. When considering items that affect the player positively (*Damage Up* and *Speed Up*), they also have a low collect rate. On the other hand, among the items that affect the player negatively, the *Control Reverse* and *Speed Down* items had the lowest rating (5.26% and 3.45%, respectively) with exception of the *Damage Down*. According to Table 1, *Damage Down* is the second most spawned item with a high collecting rate.

In order to understand the influence of both positive and negative effects on the game sessions, we extract the correlation (using the Spearmans ρ) between the total number of collected items of each type, the total time the participant survived, and how many times he/she hit an enemy. Each participant produced a tuple in the form $<Time, \sum HE, \sum DU, \sum DD, \sum SU, \sum HitEnemy>$ for each session, totalizing 12 tuples per session. These 12 tuples have been used for calculating the correlation for each session. Please notice that *Speed Down* and *Control Reverse* items have not been considered, as just one of these items has been collected by the participants during the whole experiment. Table 2 presents the correlation among these variables for each session.

According to Table 2, *Healing* has a high correlation with how long the player survived and hitting enemies for all sessions. As expected, the player can survive longer and hit more enemies every time the

Table 1
Total of items spawned, collected, and rate.

	Spawned	Collected	Rate
Healing (HE)	261	124	51.34%
Control Reverse (CR)	19	1	5.26%
Damage Up (DU)	40	10	25.00%
Damage Down (DD)	46	11	23.91%
Speed Up (SU)	38	17	44.73%
Speed Down (SD)	29	1	3.45%

Table 2
Correlation between each item type in the game and player hitting enemies and time survived.

	Session 1		Session 2		Session 3	
	Hit	Time	Hit	Time	Hit	Time
Healing	0.91	0.81	0.62	0.71	0.88	0.86
Damage Up	0.53	0.59	0.58	0.69	0.39	0.59
Damage Down	0.03	0.08	−0.36	−0.47	−0.27	−0.36
Speed Up	0.55	0.42	−0.27	−0.17	0.43	0.40

player heal himself by picking a healing item. For instance, considering the mean of healing items picked ($\bar{h} = 3.70$) and survival time ($\bar{t} = 31.41$ sec.) for all sessions, participant P12 is the one who most collected *healing* items (31 collected) as well as the one who survived the longer ($P_{12}^{\bar{t}} = 73.00$ sec.).

In the same way, *Damage Up* increases the player's damage to enemies, allowing he/she to kill more enemies in the game field and contributing to the player's survival by staying alive longer (correlation of 0.69 in session 2). However it is important to observe that it does not have a high correlation with hitting enemies (maximum of 0.58 in session 2, decreasing to 0.39 in session 3) as it does not relates to accuracy. Thanks to provenance data, a game designer could opt to add a new item that enhances the player's accuracy for a specific time to increase his hit-ratio. In an opposite way, *Damage Down* reduces the player's damage to enemies and so should present a neutral or inverse relationship with player survival time and hitting enemies. When looking at Table 2, it is possible to see that in session 1 this item had almost no contribution to the player's survival time nor hitting enemies. However, in the next two sessions, it contributed to reducing the player's survival time (−0.47 in session 2) and hitting enemies (−0.36 in session 2). For instance, P10 is the one who collected the most *Damage Down* item (a total of 3), living below the mean ($P_{10}^{\bar{t}} = 20.33$ sec.).

Finally, *Speed Up* has an interesting behavior on the players session. In the first session, it presented a directed relationship with the survival time and hitting enemies (0.42 and 0.55, respectively), indicating that players would take advantages of getting this item. However, in the second session, it contributed negatively to the players survival time and hitting enemies (−0.17 and −0.27, respectively), becoming positive again in the third session. The observed analysis indicates a possible design problem with this item. Normally, it is expected that positive effects give the player more advantages in relation to obstacles, as the case observed for both *Healing* and *Damage Up*. Additionally, the provenance data shows us that while the participants have an effect that maximizes their speed, they normally tend to collect items on the scenario. P12, for instance, was the participant that most collected *Speed Up* items (4 in total) in addition to being the one that most collected *Healing* items ($P_{12}^{\bar{h}} = 10.33$). In our exit interview, we asked the participants which items they felt more beneficial and harmful to them (excluding the *Healing* item). The result is presented in Fig. 23, indicating that *Damage Up* was considered the most beneficial. On the other hand, all the three negative items were classified equally by the participants.

According to Table 2, it is possible to see that *Damage Up* was the item that most beneficiate the participants, since it has the highest correlation with both *Hit Enemies* (0.58 in session 2) and *Time* (0.69 in session 2). The results presented in Fig. 23(a) also demonstrate this perception by the participants as most of them (81.80%) chose this item. On the other hand, all the harmful items had the same distribution among the participants (33.33%), according to Fig. 23(b).

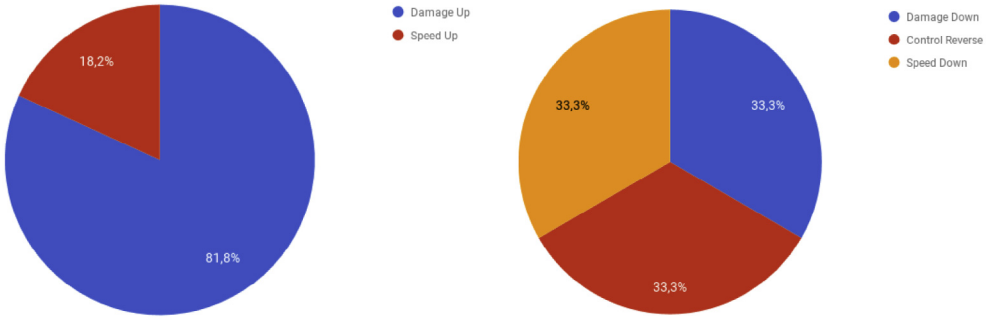
DQ4: How the participants perceive the difficulty of the game as a whole?

In order to analyze this element, we used a 10-point Likert scale in the exit interview for asking the participants about their evaluation in terms of the game level of difficulty (1 mean easy and 10 hard), enemy spawn rate (1 indicates low and 10 high), and items spawn rate (1 indicates low and 10 high).

According to Fig. 24(a), neither of the participants found the game to be easy. All of them classified the game from medium to high difficulty. Even though the majority of the participants increased their skill, as showed by the provenance data, it was not enough for the game to be classified as easy. Based on participants' characteristics presented in Fig. 17(b), most of them can be considered an experienced player. This indicates that the game should be even more challenging to grasp from novice players, requiring lowering the game difficulty to accommodate them. Thanks to the provenance analysis, it was possible to observe the same for the spawn rate of enemies (Fig. 24(b), where most of the players found it to be too high. Finally, when looking at the item spawn rate (Fig. 24(c)) the participants neither found it to be too low or too high.

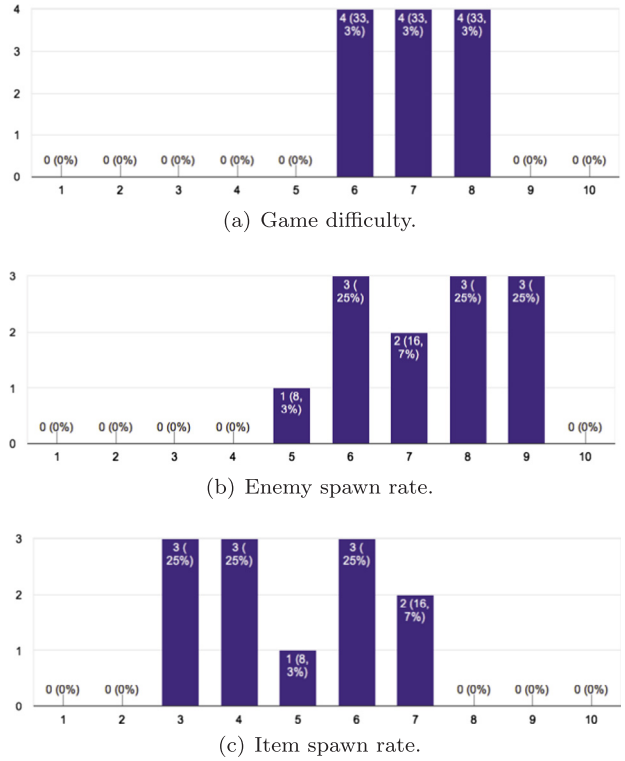
6. Conclusion

This paper presented the use of PinG for Unity, a framework for game telemetry that tracks the actions and events alongside with their cause-and-effect relationships. Our framework facilitates the process of tracking and storing the provenance data for exploration and analysis. This provenance data can aid the detection of gameplay issues, support developers for a better gameplay design, identification of game sections where players had issues and the reasons behind these issues, and mining behavioral patterns from individual sessions or groups of sessions. Moreover, we showed how PinG could be used to extract provenance data, giving examples of some common analyses. By using two open source games, we demonstrated the possibility of referencing the



(a) Most beneficial item. (b) Most harmful item.

Fig. 23. Most beneficial and harmful item according to participants.



(a) Game difficulty.

(b) Enemy spawn rate.

(c) Item spawn rate.

Fig. 24. Participants' perception about game level of difficult, enemy and item spawn rate.

provenance data in the game map to better visualize and understand the events of a game session. Besides that, we also used PinG in a game developed from scratch, collecting provenance data from 12 invited participants for analysis. The analysis has been guided by four design questions (adapted from Schell [18]) normally used during the game balancing process. Furthermore, we believe that the richness of the provenance data extracted when using the PinG for Unity framework provides the necessary means to make other types, and possibly deeper, game data analyses.

We are currently working on ways to improve the PinG framework to automate even further the data tracking, especially for influences and, in the future, possibly implement the framework for other game engines. Moreover, due to the quantity of the data extracted with PinG, we are studying techniques to improve even further the visual analysis process. These studies involve, but are not limited to, automatic graph inferences, data mining, graph reduction, multiple graph analysis to compare multiple game sessions or even cycles during a game (e.g., laps in a racing game), enabling better strategies of provenance gathering that take advantage of the games genre and type.

Finally, the collected provenance data could be used as a source for a model capable of performing automatic and dynamic difficulty adjustment, considering each player's skills individually. As we show in this paper, the collected provenance data shows players' skill improvement over 58% of the participants in different ways and rates. Additionally, when analyzing the enemies level of difficulty for the player, it is possible to observe that they are different. This represents a possible situation where the number of weaker enemies could be increased, while the strongest decreased. In addition, we collected and analyzed information that clearly shows how the items collected by the players have influence over their session time as well as the killed enemies. By considering the position and velocity of the player, these items should be put in places that could be reached by the player instead of a random position.

Acknowledgment

The authors would like to thank CAPES, CNPq, and FAPERJ for the financial support.

References

[1] M. El-Nasr, A. Drachen, A. Canossa (Eds.), Game Analytics – Maximizing the Value of Player Data, Springer Science & Business Media, London, 2013.
[2] G. Zoeller, Development telemetry in video games projects, in: Game Developer Conference (GDC), 2010.
[3] R. Hunicke, Robin, The case for dynamic difficulty adjustment in games, in: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology - -ACE '05, ACM Press, New York, New York, USA, 2005, pp. 429–433. http://dx.doi.org/10.1145/1178477.1178573. URL: http://portal.acm.org/citation.cfm?doid=1178477.1178573.
[4] A. Drachen, R. Sifa, C. Bauckhage, C. Thurau, Guns, swords and data: Clustering of player behavior in computer games in the wild, Conference on Computational Intelligence and Games (CIG), 2012, pp. 163-170. http://dx.doi.org/10.1109/CIG.2012.6374152.
[5] B.G. Weber, M. John, M. Mateas, A. Jhala, Modeling Player Retention in Madden NFL 11, in: Innovative Applications of Artificial Intelligence Conferences (IAAI), 2011.
[6] C. Pedersen, J. Togelius, G. Yannakakis, Modeling Player Experience for Content Creation, Trans. n Comput. Intell. AI in Games (T-CIAIG) 2 (1) (2010) 54–67, http://dx.doi.org/10.1109/TCIAIG.2010.2043950.
[7] G. Wallner, Play-Graph: a methodology and visualization approach for the analysis of Gameplay data, in: Foundations of Digital Games (FDG), 2013, pp. 253–260.
[8] Y.-E. Liu, E. Andersen, R. Snider, S. Cooper, Z. Popović, Feature-based projections for effective playtrace analysis, in: Foundations of Digital Games (FDG), 2011, pp. 69–76. http://dx.doi.org/10.1145/2159365.2159375.
[9] M.S. El-Nasr, T.-H. Nguyen, Glyph: visualization tool for understanding problem solving strategies in puzzle games, in: Foundations of Digital Games (FDG), 2015.
[10] S. Joslin, R. Brown, P. Drennan, The gameplay visualization manifesto: a framework for logging and visualization of online gameplay data, Comput. Entertain. 5 (3) (2007) 6, http://dx.doi.org/10.1145/1316511.1316517.
[11] J.H. Kim, D.V. Gunn, E. Schuh, B. Phillips, R.J. Pagulayan, D. Wixon, Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems, in: Human Factors in Computing Systems (CHI), 2008, pp. 443–452. http://dx.doi.org/10.1145/1357054.1357126.
[12] L. KOHWALTER, Troy; CLUA, Esteban; MURTA, Provenance in Games, in: Brazilian Symposium on Games and Digital Entertainment (SBGAMES), 2012, pp. 162–171.
[13] Data Dictionary for Preservation Metadata, Tech. rep., OCLC Online Computer Library Center & Research Libraries Group, 2005.
[14] L. KOHWALTER, Troy; CLUA, Esteban; MURTA, SDM An Educational Game for Software, in: Brazilian Symposium on Games and Digital Entertainment (SBGAMES), 2011, pp. 222–231.
[15] T. C. Kohwalter, E. G. W. Clua, L. G. P. Murta, Game flux analysis with provenance, in: Proceedings of the 10th International Conference on Advances in Computer Entertainment, vol. 8253, Springer-Verlag New York, Inc., 2013, pp. 320–331. http://dx.doi.org/10.1007/978-3-319-03161-3_23. URL: http://link.springer.com/10.1007/978-3-319-03161-3_23.
[16] L. KOHWALTER, Troy; CLUA, Esteban; MURTA, Reinforcing Software Engineering Learning through Provenance, in: Brazilian Symposium on Software Engineering (SBES), 2014, pp. 131–140.
[17] 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, vol. 9672, IPAW 2016, Springer-Verlag New York, Inc., New York, 2016, pp. 71–82. http://dx.doi.org/10.1007/978-3-319-40593-3_6.
[18] J. Schell, The Art of Game Design: A Book of Lenses, Elsevier/Morgan Kaufmann, 2008.
[19] T.C. Kohwalter, Leonardo Gresta Paulino Murta, Esteban Gonzalez Walter Clua, Capturing Game Telemetry with Provenance, in: Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), 2017.
[20] C. Fernandez-Vara, Introduction to Game Analysis, Routledge, 2014.
[21] M. Black, R.J. Hickey, Maintaining the performance of a learned classifier under concept drift, Intell. Data Anal. 3(6) (1999) 453–474. http://dx.doi.org/10.1016/S1088-467X(99)00033-5. URL: https://www.sciencedirect.com/science/article/pii/S1088467X99000335?via%3Dihub.
[22] D. Charles, M. Black, Dynamic player modelling: A framework for player-centered digital games, in: International Conference on Computer Games: Artificial Intelligence, Design and Education, 2004, pp. 8–10. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.579.6671.
[23] T.J.W. Tijs, D. Brokken, W.A. IJsselsteijn, Dynamic Game Balancing by Recognizing

Affect, Springer, Berlin, Heidelberg, 2008, pp. 88–93. http://dx.doi.org/10.1007/978-3-540-88322-7_9. URL: http://link.springer.com/10.1007/978-3-540-88322-7_9.

[24] R.J.V. de Medeiros, T.F.V. de Medeiros, Procedural Level Balancing in Runner Games, 2014 Brazilian Symposium on Computer Games and Digital Entertainment, IEEE, 2014, pp. 109–114, , http://dx.doi.org/10.1109/SBGAMES.2014.30 http://ieeexplore.ieee.org/document/7000038/.

[25] J.K. Olesen, G.N. Yannakakis, J. Hallam, Real-time challenge balance in an RTS game using rtNEAT, in: 2008 IEEE Symposium On Computational Intelligence and Games, IEEE, 2008, pp. 87–94. http://dx.doi.org/10.1109/CIG.2008.5035625. URL: http://ieeexplore.ieee.org/document/5035625/.

[26] L.J.F. Perez, L.A.R. Calla, L. Valente, A.A. Montenegro, E.W.G. Clua, Dynamic game difficulty balancing in real time using evolutionary fuzzy cognitive maps, in: 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), IEEE, 2015, pp. 24–32. http://dx.doi.org/10.1109/SBGames.2015.17. URL: http://ieeexplore.ieee.org/document/7785838/.

[27] Y. Gil, S. Miles, PROV Model Primer, 2010.