Player Behavior Profiling through Provenance Graphs and Representation Learning

Sidney Araujo Melo
Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brazil
sidneymelo@id.uff.br

Troy C. Kohwalter
Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brazil
troy@ic.uff.br

Esteban Clua
Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brazil
esteban@ic.uff.br

Aline Paes
Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brazil
alinepaes@ic.uff.br

Leonardo Murta
Universidade Federal Fluminense
Niterói, Rio de Janeiro, Brazil
leomurta@ic.uff.br

ABSTRACT

Arguably, player behavior profiling is one of the most relevant tasks of Game Analytics. However, to fulfill the needs of this task, gameplay data should be handled so that the player behavior can be profiled and even understood. Usually, gameplay data is stored as raw log-like files, from which gameplay metrics are computed. However, gameplay metrics have been commonly used as input to classify player behavior with two drawbacks: (1) gameplay metrics are mostly handcrafted and (2) they might not be adequate for fine-grain analysis as they are just computed after key events, such as stage or game completion. In this paper, we present a novel approach for player profiling based on provenance graphs, an alternative to log-like files that model causal relationships between entities in game. Our approach leverages recent advances in deep learning over graph representation of player states and its neighboring contexts, requiring no handcrafted features. We perform clustering on learned nodes representations to profile at a fine-grain the player behavior in provenance data collected from a multiplayer battle game and assess the obtained profiles through statistical analysis and data visualization.

CCS CONCEPTS

• **Applied computing → Computer games**; • **Computing methodologies → Cluster analysis**; *Neural networks.*

KEYWORDS

player behavior profiling, cluster analysis, graph representation learning

ACM Reference Format:
Sidney Araujo Melo, Troy C. Kohwalter, Esteban Clua, Aline Paes, and Leonardo Murta. 2020. Player Behavior Profiling through Provenance Graphs and Representation Learning. In *International Conference on the Foundations of Digital Games (FDG '20), September 15–18, 2020, Bugibba, Malta.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3402942.3402961

1 INTRODUCTION

Game Analytics has become a relevant area in recent years due to its impact on both the game industry and game research environment [13]. In [10], the authors remark that Game Analytics has emerged as an umbrella term for the various data scientific endeavors in industry and academia. They also stand that shared definitions and terminologies must be better formalized due to the recent establishment of the field. For them, Game Analytics fundamentally deals with the use of data analysis for game production, game performance, and—crucially—player behavior understanding [10]. The outcomes of Game Analytics tasks, such as game balancing [2][37], *player profiling* [28][35], and detection of failures during game design [40][16], are employed to enhance the game development process or the game experience itself. In this context, Hooshyar et al.[20] points out that profiling and predicting player behavior is of the utmost importance when developing games.

A necessary condition for any Game Analytics task is the existence of recorded game session data. Through telemetry, it has become trivial to track and remotely gather game session data from players. There are different strategies for gathering and storing data collected from games, ranging from raw log-like files to more structured formats, such as game provenance graphs and the PlayGraph. Regardless of format or structure employed in these gameplay recordings, it is possible to extract several metrics of various types, including *gameplay metrics*, which are important to evaluate game design, user experience[13], and player behavior profiling.

Hooshyar et al.[20] present an overview of methods employed in player profiling, especially in an unsupervised learning setting, where the player behavior is profiled from a set of features without a "ground truth" label or previous expert annotation. However, to the best of our knowledge, the literature so far has relied on clustering gameplay metrics [20]. This characteristic might lead to two main limitations: (1) gameplay metrics are handcrafted features[1], input features that are manually designed towards a task, such as completion time or the number of enemies killed in the case of

[1]Handcrafted features is a term, also known as resource engineering, for situations where the designer need to manually define the features to be used in machine learning.

player behavior profiling; (2) since gameplay metrics are computed after key events, such as level or game completion, the player profile might not be suitable for fine-grain analysis. We tackle these limitations by combining representation learning[14], a set of machine learning techniques capable of learning features, with game provenance graphs, a data structure for recorded game session data that models causal relationships between game elements and events.

Thus, we present in this paper a novel approach for player profiling combining structured game session data, recorded as game provenance graphs, and representation learning techniques. We do that in such a way that all recorded game session data can be used, instead only the resulting gameplay metrics. In a nutshell, we use game provenance graphs as input to a graph representation learning algorithm that learns vector representations for each game element and submit player's vector representations to clustering algorithms. Resulting clusters are interpreted and labeled according to their underlying subgraph patterns.

By using a structured and/or relational representation, one can naturally handle objects, entities, characters, their properties, and their relationships in a game. With that in mind, provenance graph techniques were successfully adapted to record into a *Game Provenance Graph* the game session history, composed by the elements of the game and the causal relationships among them [24][26][25].

Recently, machine learning applied to graph-structured data has gained much attention due to advancements in deep learning, and several algorithms and frameworks that have been developed to deal with this structure [32][31][15][8][17]. In [29], a framework for edge detection in Game Provenance Graphs has been proposed using graph-based representation learning techniques. Graph representation learning methods induce a mapping that embeds nodes, edges, or the entire (sub)graphs, taking into consideration the underlying graph structure, as points in a low-dimensional vector space[18] called embeddings, which can, therefore, be used as input to a downstream machine learning method for tasks such as classification, regression, or clustering[1].

In our approach, we use the PingUMiL framework to learn representations in an unsupervised setting so that each player node representation embeds the player state and its neighboring context jointly. After that, we feed the learned representation to clustering algorithms in order to obtain player node clusters. We rely on the previous literature to choose suitable clustering algorithms for player node representations. Based on the characteristics of each technique combined in our approach, we believe that the resulting clusters are, essentially, similar subgraphs across game session that correspond not only to similar player behavior (actions and states) but also similar contexts, so that they could provide insights about player behavior at a fine-grain level. Our approach also contributes with a visualization of graph-based relationships between provenance nodes as an aid to analysis of player behavior.

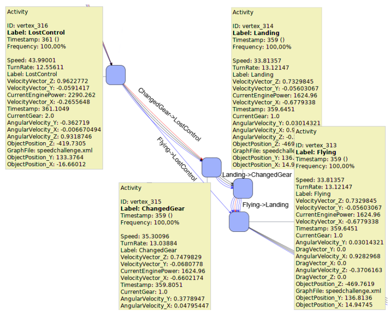
We evaluate our approach with an experiment devised to observe and discuss actual player behavior profiles obtained through our proposal's application. We conducted this experiment by applying the provenance gathering to a multiplayer flight battle game called "Smoke Squadron", using the PingUMiL framework in order to achieve node representations for all recorded game sessions. For a better comparison, we submitted the learned representations to two clustering algorithms: K-means and spectral clustering. Both methods are well-established clustering algorithms and leverage different aspects of the data distribution, which lead to different insights over the data. Then, we interpret the resulting clusters by performing a thorough analysis of the game provenance graphs using descriptive statistics on player nodes attributes and graph visualization tools. Finally, we compare and discuss obtained profiles through K-means and spectral clustering. Results show that our approach is capable of generating relevant player behavioral profiles in a strategy or tactical level by leveraging node attributes related to game mechanics and the context in which the player uses this mechanics.

The remaining of the paper is organized as follows: Section 2 presents background about Provenance in Games, Representation Learning on graphs, and clustering algorithms. Section 3 formalizes our methodological proposal. Section 4 presents the experimental setup and is further divided into two subsections: Smoke Squadron and Dataset. Section 5 presents, analyzes, and discusses our experimental results. Section 7 briefly discusses and compares our work with previous works on player behavior profiling in the literature. Finally, Section 8 concludes this work and discuss possible future works.

2 BACKGROUND

Here we briefly present the main areas that we rely on to build our proposal, namely, an overview of game provenance graphs, graph representation learning, the PingUMiL framework for representation learning on game provenance graphs, and, finally, clustering algorithms.

2.1 Provenance in games



Activity

ID: vertex_316
Label: LostControl
Timestamp: 361 ()
Frequency: 100,00%

Speed: 43.99001
TurnRate: 12.55611
Label: LostControl
VelocityVector_Z: 0.9622772
VelocityVector_Y: -0.0591417
CurrentEnginePower: 2290.262
VelocityVector_X: -0.2655648
Timestamp: 361.1049
CurrentGear: 2.0
AngularVelocity_Y: -0.362719
AngularVelocity_X: -0.006670494
AngularVelocity_Z: 0.9318746
ObjectPosition_Z: -419.7305
GraphFile: speedchallenge.xml
ObjectPosition_Y: 133.3764
ObjectPosition_X: -16.66012

Activity

ID: vertex_314
Label: Landing
Timestamp: 359 ()
Frequency: 100,00%

Speed: 33.81357
TurnRate: 13.12147
Label: Landing
VelocityVector_Z: 0.7329845
VelocityVector_Y: -0.05603067
CurrentEnginePower: 1624.96
VelocityVector_X: -0.6779338
Timestamp: 359.6451
CurrentGear: 1.0
AngularVelocity_Y: 0.03014321
AngularVelocity_X: 0.9
AngularVelocity_Z: -0.
ObjectPosition_Z: -469
GraphFile: speedchalle
ObjectPosition_Y: 136.
ObjectPosition_X: 14.9

Activity

ID: vertex_315
Label: ChangedGear
Timestamp: 359 ()
Frequency: 100,00%

Speed: 35.30096
TurnRate: 13.03884
Label: ChangedGear
VelocityVector_Z: 0.7479829
VelocityVector_Y: -0.0680778
CurrentEnginePower: 1624.96
VelocityVector_X: -0.6602174
Timestamp: 359.8051
CurrentGear: 1.0
AngularVelocity_Y: 0.3778947
AngularVelocity_X: 0.04795447

Activity

ID: vertex_313
Label: Flying
Timestamp: 359 ()
Frequency: 100,00%

Speed: 33.81357
TurnRate: 13.12147
Label: Flying
VelocityVector_Z: 0.7329845
VelocityVector_Y: -0.05603067
CurrentEnginePower: 1624.96
VelocityVector_X: -0.6779338
Timestamp: 359.6451
CurrentGear: 1.0
AngularVelocity_Y: 0.03014321
DragVector_Y: 0.0
AngularVelocity_X: 0.9282968
DragVector_X: 0.0
AngularVelocity_Z: -0.3706163
DragVector_Z: 0.0
ObjectPosition_Z: -469.7619
GraphFile: speedchallenge.xml
ObjectPosition_Y: 136.8136
ObjectPosition_X: 14.94745

Figure 1: Subgraph from a racing game.

The adoption of data provenance in the context of games was first proposed by Kohwalter et al.[23] through the PinG (Provenance in Games) framework. Following the Open Provenance Model [30], the authors defined a mapping between game elements and each type of node of a provenance graph. Players, enemies, and NPCs (Non-playable characters) are mapped as *Agent* nodes; items, weapons, potions, static obstacles, or any other object used in the game are mapped as *Entity* nodes; and actions and events are mapped as *Activity* nodes. Causal relationships between game elements are mapped as edges connecting their respective nodes, resulting in a game provenance graph. Causality indicates a relationship between two events, where the former event affects the later. The provenance approach capture causal relationships explicitly defined by the game developer. Each edge captured through the provenance approach represents a type of relationship (that can also be causal) between game objects' actions and/or states. Note that provenance allows that edges represent interactions between different game elements. An implementation of PinG is available for the Unity game engine as a plugin called PinGU[26] (Provenance in Games for Unity).

The most important advantages of provenance graphs are the modeling of causal relationships, which structures the provenance elements into a graph, and its richness of detail, i.e., the data collected at fine-grain. Figures 1 and 2 present examples of a racing game provenance graphs and their components. The graph is plotted using the player node's coordinates X and Z within the game space so that it is possible to have a general view of the race course and, for that specific scenario, follow the player trajectory and other events by traversing the graph.

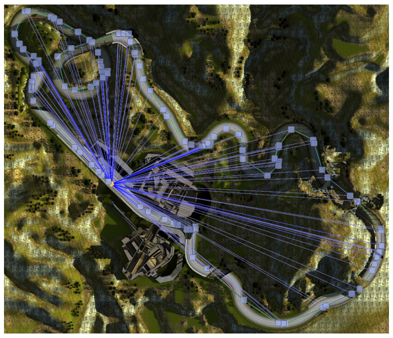


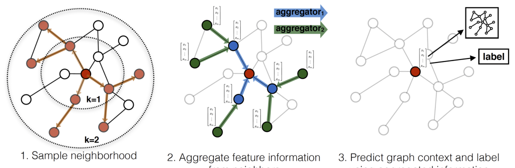
Figure 2: Racing game provenance graph example.

Figure 1 shows all the attributes of a car controlled by the player during the *Flying*, *Landing*, *ChangedGear*, and *LostControl* activities. While PinGU implements several methods to facilitate provenance capture, game developers must write domain-specific provenance tracking functions and attach them to each entity in the game [25]. Therefore, the amount of data gathered in a single node depends on the developer's design and his analytic choices.

In summary, the implementation of the data extraction algorithms and the events happening within a game session directly influence the amount of generated data[25]. Also, this rich and raw provenance data can be used in machine learning tasks in order to describe hidden patterns, aid the game maintenance, and help in future developments. This is one of the main assumptions of this research, which relies on recent advances in graph representation learning techniques.

2.2 Representation Learning on Graphs

Recently, a large amount of work has been produced to automatically learn representations (roughly speaking, feature vectors) directly from graph-structured data with machine learning [15][8][17]. Most methods in this area seek to make predictions, discover patterns, or classify nodes by transforming graph-structured data into feature information [27][36][18]. For example, by representing molecules as graphs and learning vector representations suitable for a classification task. For so, these graph representation learning



1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

Figure 3: Visual illustration of the GraphSAGE sample and aggregate approach[17]. Source: http://snap.stanford.edu/graphsage

methods incorporate information related to the structure of the graph to yield latent features that represent some properties of the data[32][38][22][17]. The idea behind these representation learning approaches is to learn a mapping that embeds nodes, or entire (sub)graphs, as points in a low-dimensional vector space, $\mathbb{R}^d$. The goal is to optimize this mapping so that geometric relationships in this learned space reflect the structure of the original graph [18]. Vector representations resulting from mapping nodes into such a learned space are called node embeddings.

In order to properly represent the nodes of a provenance graph, it is essential that we select a method able to represent node attributes. Thus, we focus on the convolutional-based approaches that are able to tackle this issue. In general, these approaches generate a node embedding iteratively. As the first step, the node embedding is initialized with the values of the node's features. At each iteration, the node's embeddings aggregate their neighbors' embeddings, generating new embeddings. These approaches determine the node embedding according to its surrounding neighborhood attributes. Therefore, they are also called neighborhood aggregation methods. Examples of these methods are Graph Convolutional Networks (GCN) [22], Column Networks [33], and GraphSAGE [17].

Figure 3 illustrates the approach implemented by GraphSAGE. Its main goal is to learn useful representations by aggregating features from a node's local neighborhood iteratively and then use graph-based loss function to fine-tune weight matrices and aggregation functions' parameters. This graph-based loss function depends on the chosen setting. In the unsupervised setting, GraphSAGE enforces similarities on representations of nearby nodes, i.e., the unsupervised setting generates generic representations based on nodes' connectivity. In the supervised setting, GraphSAGE enforces similarities on nodes representations according to a ground truth, i.e., the supervised setting generates more task-specific representations. Finally, it provides multiple aggregator architectures, i.e., functions defined for aggregating node embeddings according to its sample neighborhood. GraphSAGE is used as representation learning technique in PingUMiL [29], a game provenance graph framework for machine learning.

2.3 PingUMiL

In this subsection, we present PingUMiL, a framework for game provenance graphs driven by machine learning tasks, using GraphSAGE as a representation learning technique. The idea behind PingUMiL is to provide preprocessing routines for provenance graphs, submit graphs to a graph representation learning algorithm, and output one of the following:

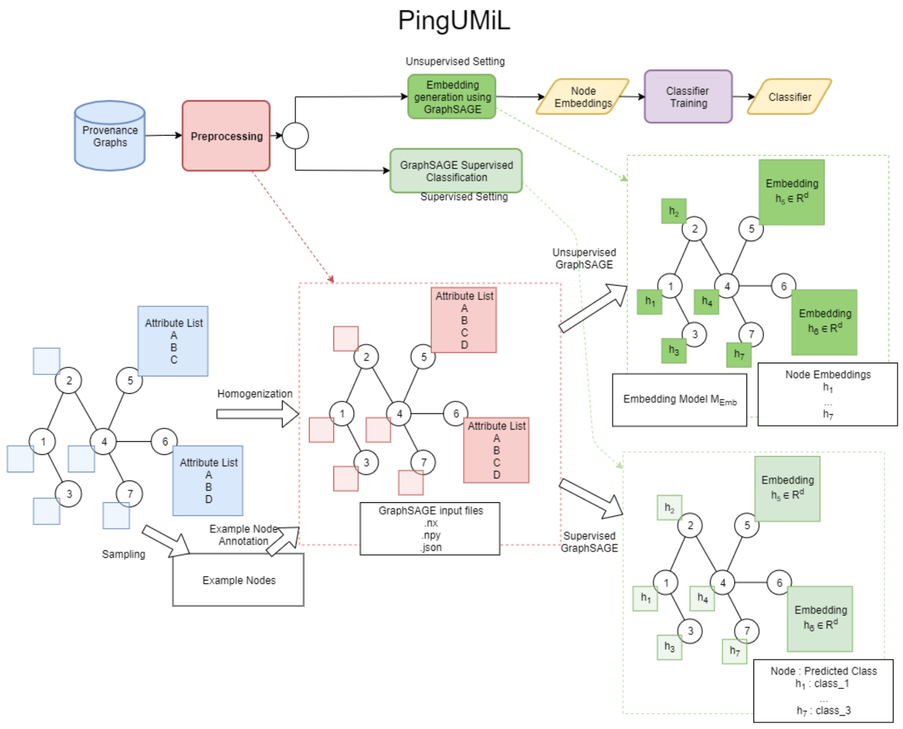


Figure 4: Overview of the PingUMiL framework.

- generic node embeddings suitable for downstream machine learning tasks, using GraphSAGE's unsupervised learning setting;
- a classification model and its input task-oriented node embeddings, using GraphSAGE's supervised learning settings;

An overview of PingUMiL is shown in figure 4. A set of provenance graphs is the input to the whole framework. These graphs must be preprocessed due to node heterogeneity, and, according to the intended machine learning task, example nodes must be defined. Particularly, example nodes are necessary for classification-based tasks; thus, it is out of the scope of this work.

As mentioned, embedding generation is realized through GraphSAGE [17], which takes as input graphs with homogeneous nodes, i.e., nodes with the same set of features. Provenance graphs with heterogeneous nodes must, therefore, be mapped into homogeneous nodes. We illustrate that process in Figure 4, where the input graph (blue) contains seven nodes of two types: (A,B,C) and (A,B,D). The output graph (pink), on the other hand, contains only one type of node: (A,B,C,D). In this case, the homogeneous node five receives all values from its heterogeneous version and a default value for the attribute D.

In a real-world scenario, a default value for an attribute should be determined by the game designer or analyst. For example, a possible default value for the *Speed* attribute is 0 if a game object does not possess a *Speed* attribute, and consequently, it does not move. Once all the nodes are homogeneous, any non-numeric attribute must be mapped into one-hot-vector representations, i.e., a k-dimensional binary vector with a single '1' value, where k is the number of possible values of the non-numeric attribute and the position of the '1' value represents the attribute value.

Algorithms for this step has been developed and requires only configurations depending on the game and its provenance graph structure [29].

Embedding generation (dark green) is made using the GraphSAGE framework, which is based on neighborhood aggregation techniques and includes several aggregation techniques based on functions and neural network architectures. GraphSAGE implements mean, LSTM, max-pooling, mean-pooling, and GCN as aggregation functions.

By applying GraphSAGE with any of the provided aggregation architectures in a supervised setting, representations are learned by trying to minimize a cross-entropy loss function, consequently trying to minimize the difference between predictions and the ground-truth. In the unsupervised setting, the algorithm minimizes a proximity function between randomly sampled neighboring nodes, where GraphSAGE tries to learn node embeddings that enforce node co-occurrences in similar contexts.

Finally, node embedding generated with GraphSAGE can be provided to a machine learning algorithm suitable for the intended task. For example, [29] performs a link detection task by encoding edges as a function of their connecting edges and training a classifier with a set of resulting encoded edges, represented in the Classifier Training step (purple). For a clustering task, however, resulting node embeddings are already suitable for clustering unless the intended clustering algorithm itself demands or suggests any preprocessing. The Classifier Training step (purple), as well as the Supervised Classification (light green) are out of the scope of this work.

3 CLUSTERING NODES EMBEDDINGS FOR PLAYER BEHAVIOR PROFILING

Cluster analysis or simply clustering is the process of partitioning a set of data objects (or observations) into subsets[19]. Each subset is a cluster, such that objects in a cluster are similar to one another, yet dissimilar to objects in other clusters[19]. To achieve that, clustering algorithms are defined by a membership function, which defines the structure of the clusters in relation to the data samples, and a search procedure, which is a strategy to cluster data given a membership function[39]. For example, according to its strategy, a clustering algorithm can be agglomerative (beginning with individual objects and merging them) or divisive (beginning with the whole set of objects and partitioning it)[4].

Clustering is useful in several exploratory pattern-analysis, grouping, decision-making, and machine-learning situations, including data mining, document retrieval, image segmentation, and pattern classification[21]. Also, clustering has been essential to player behavior profiling and, therefore, abundantly employed in several previous works[11][6][9][7].

Our proposed approach is for clustering node embeddings for player behavior profiling. In this context, we define player behavior profiling as the task of, given data that describe player states throughout gameplay sessions, extract a sub-set of this data that represent relevant patterns of player behavior, such as a trend of how the player reacts in the face of a particular threat. By using deep learning in graph representation techniques on game provenance graphs to generate node embeddings, we avoid the need for handcrafted features, such as gameplay metrics. In other words, by using graph representation techniques allows for building an end-to-end workflow for the clustering of gameplay data. As for granularity, we expect node embeddings clusters to provide insights on strategies adopted by players during the game session, such as tendencies on using a particular weapon in similar contexts, such as low life and during a boss fight.

For better understanding, we compare our approach with the one proposed by Drachen et al. [11]. The authors identify 4 cluster behaviors (Veterans, Solvers, Pacifists, and Runners) for the game *Tomb Raider: Underworld* based on cluster interpretations regarding gameplay metrics, such as causes of death, number of deaths, and completion time. In our approach, all recorded player states can be used, not only the resulting gameplay metrics. This particular work presents these four generic behaviors that characterize a player in the coarse-grain, i.e., how the player played the whole game, since gameplay metrics derive from complete playthroughs. In our approach, we would expect player behavior profiles to be more oriented to immediate strategies and playstyle.

As aforementioned, our approach is tailored towards player profiling using a structured graph related to the gameplay log, which in our case, is a provenance graph. Figure 5 presents an overview of our approach. Given a dataset of game provenance graphs, we use PingUMiL to generate node embeddings for all the nodes in the dataset. Even though we are interested in player nodes, it is necessary to embed nodes from all game objects since they also compose the context of the player state. In this context, deep learning techniques are responsible for learning node representations enforcing that player states in similar contexts must have similar node embeddings. Previous works have achieved better results by using the LSTM aggregator among all aggregators provided by GraphSAGE[17][29]. Therefore, we also adopted this approach for our representation learning stage.

After generating the node embeddings, we select the player node embeddings, since they are related to the intended profile. The following step is to cluster these nodes using an adequate clustering algorithm from a plethora of available ones. This choice is not trivial and demands knowledge over the algorithms and the data. In [4], the authors present a set of good practices when employing clustering techniques to mine behavioral game data. The authors overview hierarchical, distribution-based, and density-based algorithms and discuss in detail the K-means (and variants), Archetypal Analysis (AA), Non-negative Matrix Factorization (NMF), and Spectral Clustering algorithms. Finally, they also state five questions to define the adequate clustering algorithms according to the analyzed data, which are:

(1) Are the data high-dimensional and/or sparse? If so, consider models tailored to sparse data (AA or NMF).
(2) What is the overall goal? To build general models of player behavior or to detect extreme behaviors (e.g., cheating, gold-farming)? For the former, consider centroid-seeking models (K-means, k-medoids); for the latter, consider models such as AA.
(3) Are the data numerical or relational? For the latter, use spectral clustering or kernel methods.
(4) Are the players tightly grouped in variance space, so that K-means might have difficulties distinguishing them? If so, consider density-based approaches that do not operate on Euclidean distances.
(5) Are the data noisy? If so, density-based methods might be appropriate as they are better adjustable.

In this work, we used these five items as a framework for selecting the appropriate algorithms for our proposal: K-means and Spectral Clustering. We use K-means as a baseline clustering algorithm and also due to its general modeling capability[4]. We opt for Spectral Clustering due to its capacity for dealing with non-convex structures and the relational structure embedded in the node representation. The authors also suggest preprocessing steps for the discussed algorithms, such as dimensionality reduction for K-means[4]. After choosing a suitable clustering algorithm, we feed the node embeddings to it in order to achieve clusters.

Finally, we analyze the resulting clusters using descriptive statistics on nodes' attributes per cluster and game provenance graphs visualization tools. In the descriptive statistics step, we first analyze categorical attributes particular to the game provenance graph: type (Agent, Activity, and Entity), label (name of the Agent, Activity of Entity), ObjectTag (the name of the game entity responsible for that node). The ObjectTag is used to determine the Agent or Entity that performs an Activity. This approach gives an overview of the profile, for example, the most common activities performed in this cluster. Then, we analyze the remaining attributes in order

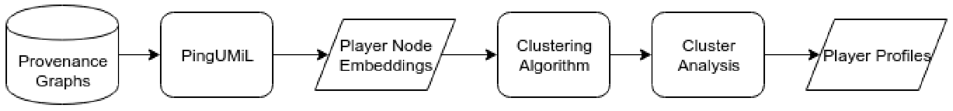


Figure 5: Overview of our approach.

to profile player behavior further; for example, how are these most common activities performed regarding attributes such as *Health Points*, *Speed*, or *Position*. Then, we use game provenance graphs visualization tools to examine the clusters as subgraphs. This last step is essential for our analysis since we can observe not only the nodes belonging to a given cluster but also how in which context that node occurs. This context could be the presence of a specific enemy, which also generated nodes present in the graph but not in the clustering.

Once the cluster analysis is complete, we condense the insights about the player behavior observed in each cluster and label player behavior profiles according to them.

In the following section, we propose an experiment to observe and discuss the player behavior profiles obtained using the described approach and assess whether the approach is capable of profiling players in a finer-grain level such as gameplay style or strategies.

4 EXPERIMENTAL SETUP

Our experiment intends to generate player behavior profiles from actual game sessions using our proposed approach and discuss its results. Our final goal is to observe the adequacy of the player behavior profiles for identifying trends on gameplay style and strategy. In order to assess our proposal, we coupled the PinGU plugin to a whole game for capturing the provenance and recording detailed information from different game sessions.

The game selected here is Smoke Squadron. It is a local multiplayer arcade flight battle game, in which the player controls a small remote airplane and battles using machine guns, missiles, and a solid smoke trail that kills at touch (Gas BTM) to create new obstacles, making every match unique, and highly dynamic [34]. Figure 6 presents a screenshot of the game.

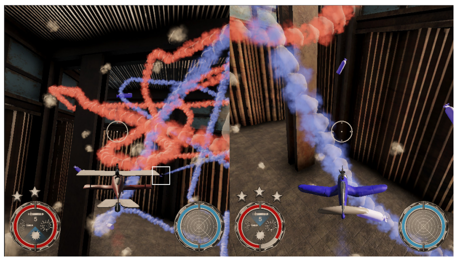


Figure 6: Screenshot of the game Smoke Squadron.

In Smoke Squadron, every player controls an airplane armed with a machine gun and three missiles: a regular missile, a smoke trail missile, and a fireworks missile, i.e., a missile that explodes after a time delay or a collision and creates smoke around its explosion point. Machine gun shots damage airplanes and solid smoke, which are destroyed if their health value goes below zero. Missiles and smoke destroy the player instantly. In order to use smoke trail and fireworks missiles, players use a smoke resource that can be recharged by picking power-ups randomly spawned in the game scenario. Beyond that, each airplane spawns a colored smoke trail, to which the spawner is invulnerable, i.e., the red smoke does not affect the red airplane. Finally, sometime after the beginning of the match, smoke clouds are randomly spawned in the environment. Player movement controls are rotation in all axes, speed boost, and air brake.

The game is planned to have three match types: 1v1, team battle (2v2), and deathmatch (all vs. all) with up to 4 players. Currently, the game has the 1v1 battle mode implemented. Also, there are two types of airplanes, differed only by their aesthetics.

The game is under development by the indie game studio *Ops! Game Studio*, which provided us access to the game source code, in order to generate a build version with provenance data capture and realize experiments with the captured data.

4.1 Dataset

Preliminary steps are required for creating the dataset of provenance graphs with Smoke Squadron game sessions' data. First, the provenance capture plugin PinGU must be instantiated into the source code of Smoke Squadron. Instantiating the PinGU in a game involves adding the provenance hooks (i.e., telemetry hooks) for capturing game objects, their actions, states and attributes, and creating relationships for these actions and states.

After instantiating the PinGU plugin into the game, game sessions must be played and recorded as provenance graphs to compose the dataset. Our dataset is composed of the game sessions produced by four participants, all male, with ages between 20 and 28, with no prior knowledge of the game. The players were recruited locally to take part in the capture process and the experiment was explained in details before the game sessions. All game provenance graphs are anonymous since they contain no identification attributes and, therefore, preserve the privacy of the players. The gameplay capture lasted three hours and generated 37 provenance graphs, totaling 168 thousand nodes and 239 thousand edges. The average number of nodes per graph is 4,427, and the average match duration is 185 seconds.

For the sake of conciseness, we list below the attributes that a node from a Smoke Squadron provenance graph might contain:

- Type attribute: Provenance node type, i.e., Agent, Activity, or Entity.
- ObjectTag attribute: determines the game element responsible for instantiating that node. The possible values are:
  - Player01, Player02: refers to a Player's Agent and Activity nodes.
  - Smoke: refers to Smoke's Activity and Entity nodes.

  - Rocket: refers to Rocket's Activity and Entity nodes.
  - SmokeItem: refers to SmokeItem's Entity nodes.
  - SmokeItemHalf: refers to SmokeItemHalf's Entity nodes.
  - SmokeSpawner: refers to ItemBox Agent node.
- Label attribute: determines a label for the node. The possible values depend on the Type and ObjectTag values. For example, the label value of a node with ObjectTag Missile and Type Entity is the type of the missile; on the other hand, the label value of a node with the same ObjectTag but with Type Activity is the action performed by that missile.
- Health related attributes: health points and the number of lives.
- Movement related attributes: position, direction and rotation in axis X, Y and Z, and Speed.
- Input related attributes: Input on angle rotation, acceleration and braking triggers.
- Physics related attributes: Throttle, Engine Power, Rotation effect ratios.
- Weapon related attributes: Weapons and Smoke's counters, cooldowns and explosion timers.

5 RESULTS AND DISCUSSION

In this section, we present and discuss the results of our experiment. First, we seize the opportunity to show how the node embeddings were distributed in the learned vector space. Then, we present the clusters obtained by the previously chosen clustering algorithms, namely K-means and spectral clustering, and label them according to the node attributes statistics and the subgraph visualization. Finally, we compare the obtained profiles and discuss them regarding the main assumptions and threats to the validity of this work.

5.1 Node Embeddings and Learned Vector Space

As aforementioned, we use PingUMiL to embed nodes into vector representations called node embeddings. One of the main assumptions of our work is that node embeddings learned by PingUMiL reinforces contextual similarity among nodes by learning a vector space. Therefore, we expect that player and other game elements' node embeddings occupy different subspaces in the learned vector space.

In our experiment, we feed the Smoke Squadron dataset to PingUMiL.LSTM and retrieve node embeddings for each node, including the player nodes, which are the ones we are most interested in this work. Embedding generation step for this experiment took on average 40 seconds, using a GeForce 940MX GPU. In order to assess the previously mentioned assumption, we apply dimensionality reduction on the node embeddings using PCA, as shown at the scatter plot in Figure 7(a). We observed that player nodes were densely grouped into the rightmost section of the learned vector space while the other game elements (Smoke, Missiles, Items) are widely distributed in the leftmost section of the space.

Notice that an intersection between the two groups of points exists. This intersection shows GraphSAGE's strategy of reinforcing similarities between nodes in similar contexts. Examples of this phenomenon are the Player and Smoke objects. Even though these objects are different, they receive damage from the same type of weapons. For this reason, it is expected that some Smoke nodes' embeddings are closer to Player nodes' embeddings.

We understand that result as strong evidence of node embedding's representational power, especially in terms of contextual similarity. The next step is to cluster the Player nodes, i.e., the Player points in the learned vector space, obtaining the player profiles based on the node attributes and their contexts.

5.2 K-means

By submitting the player nodes to the K-means clustering algorithm with $k = 5$, we obtain five player node clusters that can be observed in Figure 7(b). The choice for the k parameter in the algorithm is arbitrary. In this experiment, we found that five clusters could represent five generic and differentiated behaviors manifested by the players along game sessions, even though a higher number of clusters would add more specificity in the cluster semantics.

The interpretation process of the clusters followed two steps: statistical analysis of the attributes per cluster and graph visualization. In the statistical analysis, we first analyzed the categorical attributes, followed by the numerical attributes. This analysis gives insights on the main actions taken and how the player performed them for that cluster. In the graph visualization step, we colored player nodes on the game sessions graphs using the same colors in Figure 7(b). Besides, the non-player activity nodes (a missile spawning smoke or a smoke cloud taking damage, for example) are colored in black, and smoke nodes entities are colored in grey (nodes representing the moment these clouds have spawned). By visualizing the graph, we can directly observe both the player status and the context to which that cluster refers. Finally, we label each cluster with a single word that we believe that characterize the behavior represented in that cluster.

Using the insights provided by this interpretation process allowed us to label the five clusters as five behaviors: (0) Hasty, (1) Rogue, (2) Killer, (3) Average, (4) Fighter.

Cluster 0 represents the "Hasty" behavior. This cluster contains 31.92% of all damage nodes, which leads us to a conflict-driven behavior. Damage nodes represent that the current player was damaged. Nodes with offensive behavior such as shooting or firing missiles are observed in the cluster, but not as much as in the "Killer" behavior. On the other hand, all the nodes representing collision with Smoke elements belong in this cluster, as well as 36.62% of all the nodes representing collision with the wall. These collisions could be the result of both offensive and defensive hasty maneuvers during conflicts. Since Smoke is a central resource for attacks, we observed that attributes *SmokeCapacityCounter* and *SmokeCooldownCounter* present the least skewed distributions among all clusters, which implies that nodes from this cluster are distributed along several values of this resource consumption. The resource is consumed, for example, by using an attack such a Fireworks Missile. Also, nodes from this clusters have Cooldown values above 0 seconds, which corroborates that a weapon was fired. These observations point out that this behavior frequently occurs during or after a conflict. The cluster also contained 73.39% of all nodes regarding Item collection, i.e., the player recharged their Smoke resources. Through visualization, we noticed that this behavior is related to bad choices on strategies that might end up on defeat. In 8(d), the

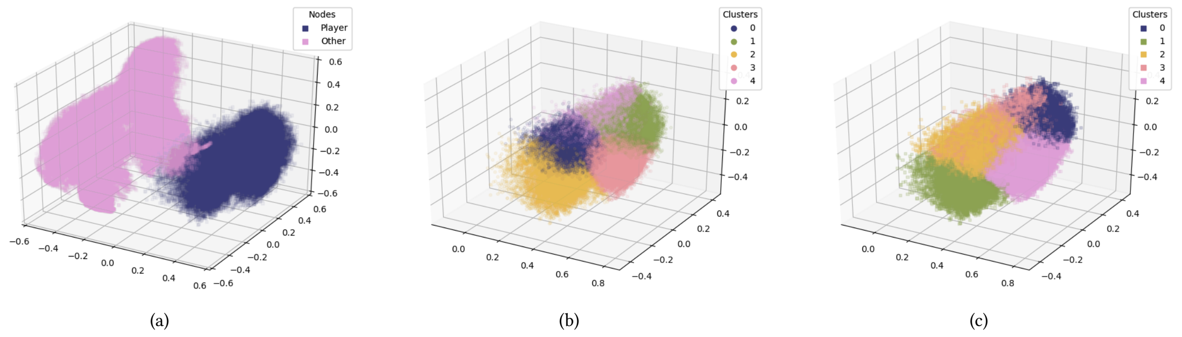


Figure 7: Scatter plots of node embeddings. (a) Scatter plot of all node embeddings generated with PingUMiL.LSTM. (b) Scatter plot of player node clusters with K-means (k=5). (c) Scatter plot of player node clusters with Spectral Clustering (k=5).

Hasty behavior was attributed to the bottom player when he or she fired a Smoke Missile and stayed in its neutral Smoke cloud trail, leading the player to a death node.

Cluster 1 represents the “Rogue” behavior. This cluster is characterized by the high speeds (using the acceleration trigger) and low positions on the Y-axis taken by the player. By flying close to the ground, the player makes it harder for the adversary to shoot effectively due to the Smoke elements that tend to spawn and float in the middle section of the scenario. Since the lowest section contains less floating Smoke clouds, it is safer to fly at higher speeds. In 8(e), both players fly freely for most of their subgraphs. It is possible to notice that the bottom player flies around the map at high velocity due to the small increase of sparsity in his nodes.

Cluster 2 represents the “Killer” behavior. This cluster contains 85.94% of all Respawning nodes and 77,11% of all Death nodes, i.e., these clusters contain effective conflict-related nodes. Respawning and Death nodes represent that the current player respawned or died, respectivelly. With 74.08% of all Fired Missile nodes and 75.78% of all nodes registering the use of the air brake trigger, i.e., this behavior indicates lower speed for better aim and, consequently, effectiveness in the conflict. Also, this cluster presented a trend of higher positions on the Y-axis compared to the others, reinforcing that conflicts tend to occur on higher sections of the scenario. In Figures 8(a), 8(b), and 8(c), we have a sequence of subgraphs from the same game session. First, both players maneuver to face each other, as can be seen in Figure 8(a), which suggests being attributed as Killer behavior. Next, the bottom player slows down, using the Air Brakes triggers. Notice that in the same period, the bottom player traveled less than the top player. Finally, the bottom player fires a Smoke trail missile in the trajectory of the adversary, which collides with the trail and dies. We believe that the concentration of Death nodes in this cluster drives the algorithm towards finding the most lethal behaviors. However, we observed that some single Killer nodes are scattered throughout the graphs due to Death nodes, such as in Figure 8(d). We understand that Death nodes are very similar, hence the concentration on these clusters, but the actions performed by the player that trigger the Death nodes differ, hence not every offensive action belongs to the Killer behavior.

Cluster 3 represents the “Average” behavior. This cluster contains 54.93% of all collisions with walls and 43.44% of all machine gun shooting. Positions on the Y-axis are similarly distributed to cluster 2, and SmokeCapacityCounter is at its highest value, i.e., the player is completely loaded for shooting missiles. However, observing the generated visualization, this behavior seems to relate to maneuvers before and after a conflict. In Figures 8(b) and 8(c), the top player changes its direction when the bottom player slows down. In Figures 8(d) and 8(e), this behavior precedes Hasty and Fighter behaviors.

Cluster 4 represents the “Fighter” behavior, which is very similar to the “Hasty” behavior. The difference lies in the existence of 22.89% of all Death nodes in this cluster, resulting from Missiles and Shootings and not from collisions with Smoke or Walls. In Figure 8(e), the player with the rightmost subgraph is attributed to the Fighter behavior just before engaging the Rogue adversary. In Figures 8(b) and 8(d), it is also possible to notice that nodes attributed with the Fighter behavior are usually close to Average and Killers, especially if the player is offensively successful in the latter.

Figure 8(f) allows seeing a number of small behavior subgraphs. This phenomenon seems to be more frequent in conflicts where the players do not interact with other game elements, especially with each other. It is possible that such nodes do not contain useful information for profile interpretation; however, due to the K-means algorithm and its parameters on this experiment, these nodes ended up assigned to clusters. This phenomenon is expected and is the reason why we take these results as a baseline for our approach and intend to compare it with a more powerful clustering algorithm, namely, Spectral Clustering.

5.3 Spectral Clustering

After generating five profiles using K-means, we developed a new clustering using Spectral Clustering method. Similarly, we generate five clusters in order to compare the resulting profiles. The interpretation process is the same as presented in the K-means experiment. We list below the resulting five behaviors using the identification number from Figure 7(c), which we label to the best of our interpretation: (0) Bird, (1) Sniper, (2) Hasty, (3) Victim and (4) Stalker.

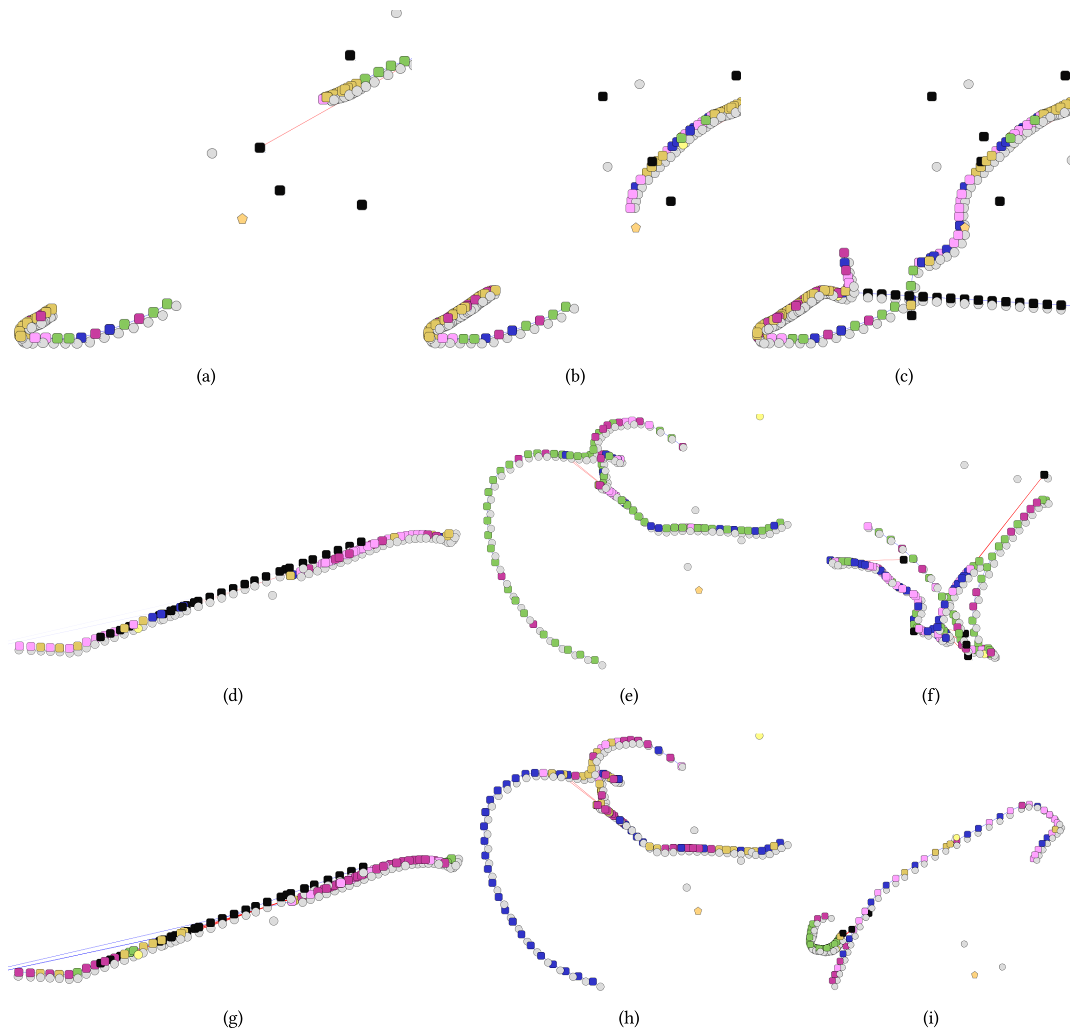

(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 8: Visualization of Smoke Squadron's game provenance graphs, where the nodes get the colors of their clusters. All figures represent different situations along game sessions.

Cluster 0 represents the "Bird" behavior. This cluster occupies the same region as the Rogue behavior; therefore, they have characteristics in common: high speeds (using the acceleration trigger) and low positions on the Y-axis. However, the Bird behavior contains fewer nodes than the Rogue, especially conflict-related nodes. In Figure 8(h), we see a long subgraph of Bird behavior by the bottom player. It is also possible to compare this result to the Rogue behavior in 8(e) and notice that this clustering performed more consistently in this example.

Cluster 1 represents the "Sniper" behavior and occupies the same region as the Killer behavior. Similar to the Bird, the Sniper is smaller than its K-means counterpart. Different from the Killer, the Sniper contains almost no Respawning nodes and Death nodes. The similarity between Killer and Sniper lies in the 41.09% of all Fired Missile nodes and 56.45% of all nodes registering the use of the air brake trigger and the trend of higher positions on the Y-axis. However, the ratios of the aforementioned nodes are at least 30% smaller than in Killers. In Figure 8(g), the leftmost player is a Sniper, which is characterized for slowing down his movement to turn around and fire a missile towards his adversary.

Cluster 2 represents a broader "Hasty" behavior. Different from Bird and Sniper, this behavior cluster is more extensive than its K-means counterpart. This cluster contains 60% of all damage nodes, all nodes representing collision with Smoke elements, 38.73% of all nodes representing collision with the wall, attributes *SmokeCapacityCounter* and *SmokeCooldownCounter* present the most uniform distribution of all Spectral clusters, 86.72% of all nodes regarding Item collection, i.e., the same characteristics from K-means' Hasty Behavior. Also, the ratio of nodes containing active acceleration triggers is 10% larger than K-means' Hasty. In 8(g), the player with the leftmost subgraph has the Hasty behavior just before colliding with a Smoke cloud, similar to its K-means counterpart 8(d).

Cluster 3 represents the "Victim" behavior. This cluster contains 90.36% of all Death nodes and 29.63% of all Fired Missiles nodes. In this particular case, we observe in Figure 7(c) that cluster 3 is distributed sparsely across cluster 2. We believe that this effect is a consequence of a grouping focused on the Death nodes. At first sight, this cluster, even though separating a very relevant group of nodes, does not seem to provide insights on player behavior and mostly fills the gaps between other profiles, as observed in Figures 8(g), 8(h), and 8(i).

Cluster 4 represents a broader "Stalker" behavior. This cluster contains 61.27% of all collisions with walls and 52.88% of all machine gun shooting, which are higher ratios than in its K-means counterpart. Similarly, positions on the Y-axis are equally distributed to the Sniper, and SmokeCapacityCounter is at its highest value, i.e., the player is completely loaded for shooting missiles. In Figures 8(g) and 8(h), the Stalker player engages in a conflict against his adversary.

Due to its ability to deal with non-convex structures, Spectral Clustering provided a more consistent cluster than K-means, presenting less single node subgraphs in visualizations. However, both clustering algorithms were capable of generating representative node clusters for player behavior profiling. These clusters could be used to infer non-observed player behavior into profiles, allowing the developer to know profiles of the players and how the player plays, reinforcing or reviewing game mechanics for specific audiences.

6 THREADS TO VALIDITY

It is important to address threats to the validity of this experiment. First, as construction threats to validity, the node homogenization step in the PingUMiL framework inserts default values for nonexistent attributes to a given node. These default values are chosen arbitrarily and might bias the embedding process. Also, the choice for the number of clusters is arbitrary and might not reflect the optimal clustering for profile interpretation. Regarding clustering techniques, it is possible that a density-based approach could generate interesting profiles as well, since they are capable of distinguishing relevant date from noisy data.

As external threats to validity, the presented experiment's dataset is composed by matches of four players that belong to a very restrict demographics. This could also bias the results. Furthermore, a dataset with more player with different playstyles could also provide more generic profiles than the ones observed in the experiment.

7 RELATED WORK

In this section, we compare and discuss our approach with similar works in the literature around three key points: input data for each method, use of representation learning, or any other data transformation method and clustering algorithms.

In [11], [12], [3], authors use gameplay metrics such as kill/death ratio, character level, and scores as input data. [3] also combines metrics with geographical positioning to generate spatio-temporal profiles. [7] uses sequences of behavioral events that were translated into behavioral metrics. On our approach, we use as input data the raw provenance graphs with the whole game session history, without feature engineering or metrics computation. Finally, we employed K-means and Spectral Clustering.

In [11], the authors use Emergent Self-Organizing Maps (ESOM), a type of neural network. The ESOM learns a mapping from input data to another dimensional space, usually in 2D. To the best of our knowledge, ESOM has been the only unsupervised neural network method applied for player behavior profiling. Our approach employs a graph-based representation learning method using LSTM to generate node embeddings followed by a clustering stage.

Finally, several clustering algorithms have been used for player behavioral profiling: K-means[5][12], Archetypal Analysis and variations [12], ESOM visualization maps[11], DEDICOM[5][3], Spectral Clustering[5]. We opted for K-means and Spectral Clustering following guidelines from [4].

8 CONCLUSION

In this paper, we propose a novel approach for player behavior profiling based on graph representation learning and clustering. Our method uses raw game provenance graphs as input data and learns embeddings for each node leveraging not only their attributes but also their neighboring context. Finally, the induced embeddings are submitted to adequate clustering algorithms following the guidelines defined in [4]. One of the main objectives of the approach is to propose an alternative for player profiling that requires no feature engineering or gameplay metrics, which was achieved using graph representation learning. We also wanted to observe whether the profiles could define player behavior throughout the game session. With that goal, we relied on visualizations tools on profiles obtained by K-means and Spectral Clustering and observed, notably for Spectral Clustering, a potential tool for understanding and classifying player behavior through its subgraphs.

As a future work, we intend to use behavioral profiles generated by our approach as classes for player behavior prediction in gameplay time. That could be achieved by generating node embeddings during the gameplay session and inferring its profile using a classifier previously trained. However, to achieve more representative profiles and better classify player, a broader and more generic dataset is needed. We also intend to explore other clustering algorithms, such as density-based and hierarchical ones.

It is also crucial to state that replications of the presented approach, specially using game provenance graphs, should follow ethical practices that preserve player privacy. For our experiments, captured graphs contain no personal information of the player and, therefore, are anonymous since such information was not available. This was possible due to the local multiplayer setting of the game. However, in online games, where sensitive information such as username is available and could be associated with captured graphs, it is important to enforce privacy and anonymity. Another future work is to analyze and discuss issues concerning game provenance graph based approaches, and propose solutions for guaranteeing privacy, anonymity, and other ethical concerns.

Although we present results for a specific game, our approach can be adapted and applied to any game that collects provenance data. Finally, our approach could be used for other graph-structured problems outside digital games as well, e.g., scientific workflow or traffic simulation.

REFERENCES

[1] Ethem Alpaydin. 2009. *Introduction to machine learning.* MIT press.

[2] Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. 2005. Extending reinforcement learning to provide dynamic game balancing. In *Proceedings of the Workshop on Reasoning, Representation, and Learning in Computer Games, 19th International Joint Conference on Artificial Intelligence (IJCAI).* 7–12.

[3] Myat Aung, Simon Demediuk, Yuan Sun, Ye Tu, Yu Ang, Siva Nekkanti, Shantanu Raghav, Diego Klabjan, Rafet Sifa, and Anders Drachen. 2019. The trails of Just Cause 2: spatio-temporal player profiling in open-world games. In *Proceedings of the 14th International Conference on the Foundations of Digital Games.* 1–11.

[4] Christian Bauckhage, Anders Drachen, and Rafet Sifa. 2014. Clustering game behavior data. *IEEE Transactions on Computational Intelligence and AI in Games* 7, 3 (2014), 266–278.

[5] Christian Bauckhage, Kristian Kersting, Rafet Sifa, Christian Thurau, Anders Drachen, and Alessandro Canossa. 2012. How players lose interest in playing a game: An empirical study based on distributions of total playing times. In *Computational Intelligence and Games (CIG), 2012 IEEE conference on.* IEEE, 139–146.

[6] Zoheb H Borbora and Jaideep Srivastava. 2012. User behavior modelling approach for churn prediction in online games. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing.* IEEE, 51–60.

[7] Alessandro Canossa, Sasha Makarovych, Julian Togelius, and Anders Drachenn. 2018. Like a DNA String: Sequence-Based Player Profiling in Tom Clancy's The Division. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference.*

[8] Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. 2017. Neural Embeddings of Graphs in Hyperbolic Space. *arXiv preprint arXiv:1705.10359* (2017).

[9] Benjamin Cowley, Ilkka Kosunen, Petri Lankoski, J Matias Kivikangas, Simo Järvelä, Inger Ekman, Jaakko Kemppainen, and Niklas Ravaja. 2014. Experience assessment and design in the analysis of gameplay. *Simulation & Gaming* 45, 1 (2014), 41–69.

[10] Anders Drachen, Christian Bauckhage, and Christian Thurau. 2015. The Age of Analytics. *Transactions of Computational Intelligence and Ai in Games* (2015).

[11] Anders Drachen, Alessandro Canossa, and Georgios N Yannakakis. 2009. Player modeling using self-organization in Tomb Raider: Underworld. In *2009 IEEE symposium on computational intelligence and games.* IEEE, 1–8.

[12] Anders Drachen, Rafet Sifa, Christian Bauckhage, and Christian Thurau. 2012. Guns, swords and data: Clustering of player behavior in computer games in the wild. In *2012 IEEE conference on Computational Intelligence and Games (CIG).* IEEE, 163–170.

[13] Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa. 2016. *Game analytics.* Springer.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning.* MIT press.

[15] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 855–864.

[16] Pietro Guardini and Paolo Maninetti. 2013. Better Game Experience Through Game Metrics: A Rally Videogame Case Study. In *Game Analytics.* Springer, 325–361.

[17] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems.* 1025–1035.

[18] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[19] Jiawei Han, Jian Pei, and Micheline Kamber. 2011. *Data mining: concepts and techniques.* Elsevier.

[20] Danial Hooshyar, Moslem Yousefi, and Heuiseok Lim. 2018. Data-driven approaches to game player modeling: a systematic literature review. *ACM Computing Surveys (CSUR)* 50, 6 (2018), 90.

[21] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. 1999. Data clustering: a review. *ACM computing surveys (CSUR)* 31, 3 (1999), 264–323.

[22] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[23] Troy Kohwalter, Esteban Clua, and Leonardo Murta. 2012. Provenance in games. In *Brazilian Symposium on Games and Digital Entertainment (SBGAMES).* 11.

[24] Troy C Kohwalter, Esteban GW Clua, and Leonardo GP Murta. 2013. Game flux analysis with provenance. In *Advances in Computer Entertainment.* Springer, 320–331.

[25] Troy Costa Kohwalter, Felipe Machado de Azeredo Figueira, Eduardo Assis de Lima Serdeiro, Jose Ricardo da Silva Junior, Leonardo Gresta Paulino Murta, and Esteban Walter Gonzalez Clua. 2018. Understanding game sessions through provenance. *Entertainment Computing* 27 (2018), 110–127.

[26] Troy Costa Kohwalter, Leonardo Gresta Paulino Murta, and Esteban Walter Gonzalez Clua. 2017. Capturing Game Telemetry with Provenance. In *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames).* IEEE, 66–75.

[27] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR).*

[28] Tobias Mahlmann, Anders Drachen, Julian Togelius, Alessandro Canossa, and Georgios N Yannakakis. 2010. Predicting player behavior in tomb raider: Underworld. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on.* IEEE, 178–185.

[29] Sidney Araujo Melo, Aline Paes, Esteban Walter Gonzalez Clua, Troy Costa Kohwalter, and Leonardo Gresta Paulino Murta. 2019. Detecting long-range cause-effect relationships in game provenance graphs with graph-based representation learning. *Entertainment Computing* 32 (2019), 100318.

[30] Luc Moreau, Ben Clifford, Juliana Freire, Joe Futrelle, Yolanda Gil, Paul Groth, Natalia Kwasnikowska, Simon Miles, Paolo Missier, Jim Myers, et al. 2011. The open provenance model core specification (v1. 1). *Future generation computer systems* 27, 6 (2011), 743–756.

[31] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 1105–1114.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 701–710.

[33] Trang Pham, Truyen Tran, Dinh Q Phung, and Svetha Venkatesh. 2017. Column Networks for Collective Classification.. In *AAAI.* 2485–2491.

[34] Ops! Game Studio. 2019. Smoke Squadron - Ops! Game Studio. https://opsgamestudio.com/portfolio/smoke-squadron/.

[35] Anders Tychsen and Alessandro Canossa. 2008. Defining personas in games using metrics. In *Proceedings of the 2008 conference on future play: Research, play, share.* ACM, 73–80.

[36] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2017. Graph Attention Networks. *arXiv preprint arXiv:1710.10903* (2017).

[37] Vanessa Volz, Günter Rudolph, and Boris Naujoks. 2016. Demonstrating the feasibility of automatic game balancing. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016.* ACM, 269–276.

[38] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 1225–1234.

[39] Georgios N Yannakakis and Julian Togelius. 2018. *Artificial intelligence and games.* Vol. 2. Springer.

[40] Alexander Zook, Brent Harrison, and Mark O Riedl. 2015. Monte-carlo tree search for simulation-based strategy analysis. In *Proceedings of the 10th Conference on the Foundations of Digital Games.*