Helping developers to look deeper inside game sessions

Marco Túlio C. F. Albuquerque Geber Lisboa Ramalho Vincent Corruble*
André Luís Medeiros Santos Fred Freitas

Federal University of Pernambuco, Informatics Center, Brazil *UPMC Paris 6, France

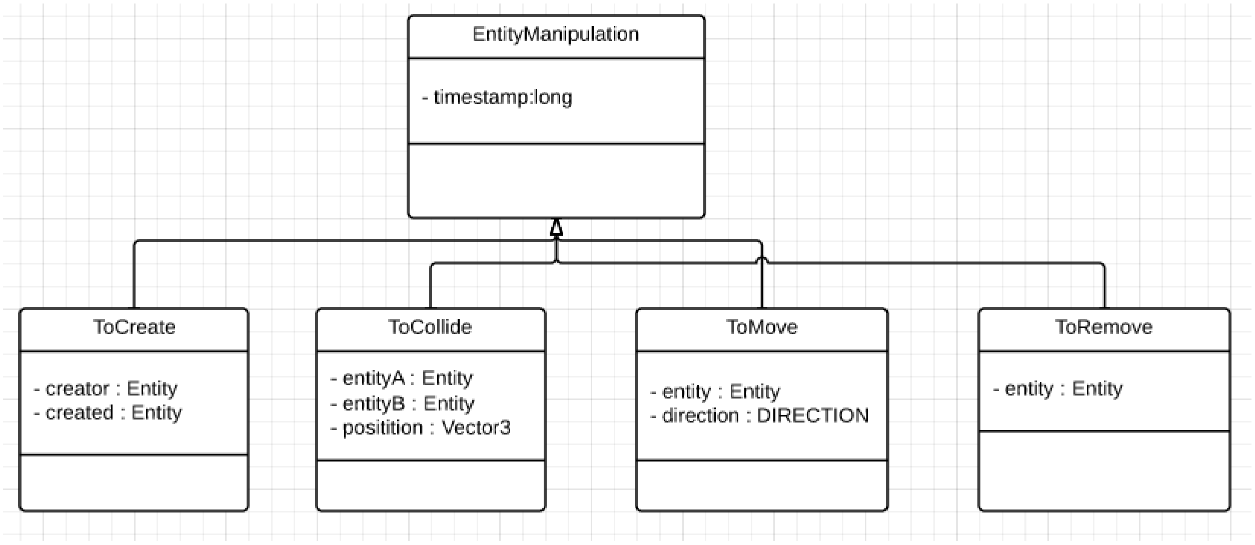

Figure 1: Mapping of the GOP ontology for the PacMan game.

Abstract

Game design and development activities are increasingly relying on the analysis of gamer's behavior and preferences data. Various tools are available to the developers to track and analyze general data concerning acquisition, retention and monetization aspects of game commercialization. This is good enough to give hints on where problems are, but not to enable a precise diagnosis, which demands fine-grained data. For this kind of data, there is not enough support or guidance to decide which data to capture, to write the code to capture it, to choose the best representation of it and to allow an adequate retrieval and presentation of it. This paper introduces GameGuts (GG), a framework devoted to give further assistance to developers in choosing, representing, accessing and presenting game sessions fine-grained data. As a case study, GG recorded sessions of a game platform with over a hundred thousand users. The logs were analyzed using a Visual Domain Specific Language (as a query language) and an ensemble of rules (as a compliance test). The results are encouraging, since we could - among other results - find bugs and catch cheaters, as well as spot design flaws.

Keywords: game analytics, knowledge representation, game data mining

Authors' contact:
`{mtcfa, glr, alms, fred}@cin.ufpe.br`
`*Vincent.Corruble@lip6.fr`

1. Introduction

Digital distribution of games is becoming mainstream for reaching a vast audience of customers [1]. In this context, game is a service [20], since the game needs to be continuously monitored and updated according to the current player's needs and behavior. Following Zynga's success, the "free to play" business model [35] (or freemium), became the industry standard [43]. Therefore, fully understanding gamer's behavior and preferences, with the use of data analysis, is an essential activity to adjust game design and correct technical flaws in order to maximize user satisfaction and, consequently, to achieve commercial success. This paradigm called "Game as a Service" (GaaS) focuses updates and analytics on three indicators of the "ARM loop": Acquisition (attracting new players), Retention (keep players engaged) and Monetization (keep users spending money in the game) [29].

Nowadays, there are effective tools adopted by the game industry to address ARM-related issues [21], which normally demand coarse-grained data, i.e. generic data such as daily active users and active revenue per user. Good to give hints on where problems are this data is not enough for a precise diagnosis. A low "revenue per user" is just as useful to understand what is wrong with the game as a fever is to a medical diagnosis. In order to go a step further in game analytics, it is necessary to acquire and analysis fine-grained data [5], i.e. the one related to the actual gameplay, such as which bullet the player used to kill the boss.

For fine-grained data, the developer has not enough support from current frameworks to decide which data to capture, to write the code to capture it, to choose the best representation of it and to allow an adequate use of it [5]. These choices are usually "ad hoc" and data granularity is not addressed with clarity. In this context, we claim that, to take full advantage of the GaaS paradigm, developers need further assistance.

Our work consists in an innovative framework devoted to help developers to define, gather, represent and visualize fine-grained game data. First, we propose a process to help the developer to choose what to gather and to represent using ontologies. Finally, we couple this process with an Application Programming Interface (API) to retrieve information using software tools, and a visual query language to improve the data retrieval, use and presentation. With respect to the state of the art, we follow a complementary approach. Data storage issues and analysis techniques are reasonably well addressed by the current tools [5], and were not an issue at any point in our work, so it is not discussed in this paper. Our focus is on providing better data acquisition and representation, using storage techniques available, to improve game analysis such as automated testing, tracking user experience and satisfaction, automatic level design, among others.

As a case study, the process is applied to games in the "Olimpiadas de Jogos e Educação" (OjE) platform. The OjE platform is a learning platform for teenagers that is used by public schools in Brazil it has over 100 000 users registered. The amount of users allowed the framework to be tested in a "real life" production environment where reliability and performance are extremely necessary. Therefore, also validating the framework regarding not only its function but also it usefulness. The results are encouraging as shown in the last section.

The rest of this paper is organized as follows. Next Section discusses gameplay data analysis. The third focus on the acquisition and representation problem. Section 4 provides the state of the art for game analytics frameworks. The GG framework is presented at Section 5. Section 6 shows how we applied the framework in a commercial game and presents the obtained results. Finally, the last section draws some conclusions and outline directions for future work.

2. Different Uses of game data analysis

In order to define the appropriate process to capture and analyze data in any domain, it is important to understand the goals of the stakeholders involved in the analysis of such data. In the game industry, there are several stakeholders, each of them with different interests.

Marketing managers are especially concerned with acquisition in terms of the performance of assets such as short teaser videos and promotional images. They are also interested in finding out which item sells the most, the price that maximizes sales, what percent of player makes a buy, what is the average revenue per active user and many more. For not being directly related to the game itself, we will not discuss marketing managers concerns are not considered in our work.

Game designers are interested in game mechanics and user experience. However, they are also interested in fine-grained data to understand precisely the problems to improve the game design [23; 24], to model opponents [25], to automate level design [26; 27] or maximize user experience [28].

Programmers and testers have special interest in bugs, since recent games had huge losses because of bugs [36] [37]. Simulating players to automatically provoke bugs may help [21], but the only reliable way to solve this problem is software testing [11]. Some games implement bug-reporting systems [30]. Programmers are also concerned with cheating. It is not only possible to cheat but also very easy [8]. Fine-grained data is required to identify active attacks (such as spoofing, replay, modification of message content and denial of service) [9].

3. Challenges in fine-grained data acquisition and representation

The focus, in this paper, is the challenges involving fine-grained data: what data to capture; how to represent the captured data using formal methods; how to code the game to provide data capture; and, the needs of a good data retrieval system.

3.1 What data to capture

Storing every possible data about each gameplay session is not an option, not only because of the required storage space or communication band, but also because of the computational tractability problems, possibly causing "the curse of dimensionality" [19]. However, deciding what data is relevant is hard (it requires lots of prior knowledge) [21]. There is then a trade-off in the choice of which data to capture not too much, not too less.

Even in fine-grained in-game data universe, there are different levels of granularity: primitive inputs (pressed keys, mouse clicks, etc.), actions (turn left, jump, etc.) and intentions (avoid an obstacle, hit an enemy, etc.) Usually, the primitive input value is too verbiage, generating intractable data. Intentions are the most abstract. For instance, different actions can "avoid an obstacle". Actions provide a better abstraction but it might lack important usability information (like, do users use the mouse or keyboard to fire the gun?). Intentions might lead to missing data since it is sometimes impossible to figure out the user action or the input used to achieve the intention.

3.2 How to represent data

The game analysis field have been using a few formalisms: attribute-value logic, first-order logic, ontologies, and DSLs. For each of them, a trade-offs regarding tractability and expressiveness must be considered.

Attribute-value logic is simple and tractable but lacks expressiveness. For instance, simple descriptions, such as Above(X,Y) ∧ Left(X,Z) ∧ Behind (X,G), concerning objects position with respect to each other,

cannot be concisely achieved using an attribute-value representation [18]. Universal and existential quantifiers are other examples of useful expressive elements that are not available. It is hard to format a query for: Is there a player that killed a certain boss without using a special ability? This would require a first-order logic representation like $\exists x,y,z$ Player(x) $\wedge$ Boss(y) $\wedge$ EspecialAbility(z) $\wedge$ Killed (x, y) $\wedge$ $\neg$ Used (x, z).
Ontologies have been used for representing game data. In terms of tractability and expressiveness, ontologies are in-between attribute value and first-order logic. The main advantage of ontologies is that they propose a representation vocabulary. This is the case of the Game Ontology Project (GOP) [3] that offers a vocabulary to represent game sessions in the action-level granularity.
DSLs can be used to represent both knowledge and reasoning on a domain. They can also use an ontology as the domain mapping stage on the creation of new DSLs [17].

3.3 How to code the game to provide data capture

In order to capture data, the developer needs to modify the game code. However, this seemingly simple task may be complicated to guarantee the separation of concerns. In other words, data capture is a crosscutting concern since it affects other aspects of the code and crosscutting concerns can generate duplicated code and excessive dependencies, increasing the code's complexity and the costs of maintenance and updates [17].
Allowing the developer to understand the capturing needs early in the development cycle improves the primary design decisions of the code and prevent issues related to crosscutting concerns. Furthermore, using a unique representation enables the standardization of the code involved in the capturing and storing of the data that also minimizes such separation of concerns issues [12].

3.4 How to retrieve and use the data

Non-technical stakeholders, like game designers, frequently need a user-friendly presentation of the game data. On the other hand, programmers prefer data in a format that can be readily used by the machine. This trade-off between human-friendly and computer-friendly format may be overcame by a clear separation of the representation language to the presentation one, which can be visual [21; 31]. It is easier to identify the paths a player takes through the gameplay and recognize errors in the design from the visual representation. Spatial game data is also commonly presented visually [13].

4. Game Analysis Frameworks: State of the art

This section defines some properties of game analytics framework that needs to be considered when creating, extending or using one that is supposed to deal with fine-grained data. Each of the properties described below is related to the challenges or trade-offs discussed in the last section.

1. Amount of Prior Knowledge Required: Most frameworks expect the game developer prior knowledge on data relevance. A framework should try to minimize this burden from the developer.
2. Expressiveness: the game data representation language must be able to express fine-grained data from sessions.
3. Separation of concerns: Coding metrics into the game, and writing its documentation, must be a simple task, and the resulting code must be easy to maintain for a long period (in the perspective of GaaS).
4. Computer-friendly format: it is important to allow that computers easily retrieve and use the stored data in order to mine information, generate content, etc.
5. Human-friendly format: data must also be presented in a user-friendly manner, visually preferably, to non-technical stakeholders.
6. Extensibility: the framework must be able to handle new metrics as the game evolves.

Flurry [7], most used analytics framework, provides all the major ARM related metrics for coarse-grained data. Regarding fine-grained data, it provides the "custom event storing format", which consists in an event name coupled with a hash table. Therefore, it is very expressive, almost anything is allowed, but it requires extensive prior knowledge since it does not provide any process or mechanism to help the developer define what to track or how to represent it. The same lack of process applies to the extensibility and computer-friendliness of this Flurry's feature. Flurry's API requires the developer to develop its own code architecture for tracking metrics not paying any attention to the separation of concerns. Flurry also provides various graphics formats to help visualize the data, being very user-friendly in the report but lacks in the information retrieval process, which is based on lists of events. Other options are available (such as Kontagent, Honey Tracks, GameAnalytics and others) but they are all very similar to Flurry. Some allow fine-grained data representation but the same criticisms concerning the lack of guidance in choosing and representing data in Flurry applies to them.
Among the academic efforts in game analytics, Memphan [6] proposes an event-driven framework that can verify compliance of events with respect to rules defined in OWL [31] with Jess [32]. The choice was to limit the language to board games that leads to a different path from the commercial frameworks. Every player action in the client is validated against a set of rules in the server. The player's actions are represented as facts on a Jess database. Although the authors found it very difficult to express all the game rules using the Jess limited expressivity, the representation language is computer-friendly and fulfill its purpose for board games. It also does not require any prior knowledge of board games designer. The authors do not mention anything about separation of concerns in the code leaving, again, the burden to the developer. In another initiative, Chan [4] creates an ontology borrowing concepts from a set of related ontologies such as the GOP, the Music Ontology [33] and others. Such an

ontology represents many concepts needed for fine-grained gameplay analysis. However, since its focus is on game quality evaluation, the framework does not address the separation of concern issues. Chan, like Memphan, is expressive and computer-friendly for its limited purposes, requires no prior knowledge. However, none of the two provides means on how to extend the representation format for other games or purposes therefore, are not extensible. Finally, the presentation of the data is not visual; in fact, both authors overlook the issue.

5. The GameGuts Framework

This section presents a series of methods and tools that aids developers in the fine-grained data management named GameGuts Framework (GG). GG consists of a detailed systematic process on how to define, represent and code fine-grained data, an API to retrieve data and, finally, a visual DSL to present the data in a human-friendly way. During this section, along with the presentation of the process, the Pac-Man - chosen for being a very popular and well-known game - will be used as an example for clarity purposes.

5.1 Main Development Choices

This sections explain GG positioning related to other frameworks according to the attributes presented in the previous section.

1. Amount of prior knowledge: Defining which analytics to track is a tedious and laborious job. Any mistake leads to missing information and inability to understand a problem. One of the main focus of GG is to reduce this necessary prior knowledge about analytics and help in the definition process, reducing the human error factor. Using the GOP as a start point drastically reduces the previous knowledge since this ontology describe nearly all possible actions in a game and has a small learning curve for game developers.

2. Expressiveness: The gameplay logs must be able to represent any action that happens in the game. GG uses GOP as a starting point. For its collaborative and open nature, GOP provides terms and vocabulary for almost any game available.

3. Separation of concerns: Most third party APIs usually forces the developers to adapt the game code in order to include analytics code. GG provides a few guidelines on how to handle the game client code using the class diagram generated when creating the gameplay representation language. As the developer follows the process of creating the representation language, the class structure necessary to integrate GG in the game code is created. This allows the developer to understand upfront which parts of the code will be affected and prevent crosscutting concerns issues.

4. Computer-friendly format: To facilitate the use of the stored data by other tools and plug-ins GG provides an API to retrieve the stored data. The Server API has the ability to retrieve individual session, specific data and scenarios from the whole set of stored sessions. GG also provide the ability to create triggers in order to process the data as it gets into the server according to the each analysis needs.

5. Human-friendly format: To overcome presentation issues of textual data and provide visual presentation such as Flurry, GG provided a detailed systematic process to help the developers create a visual query language on top of GG.

6. Extensibility: As the game evolves new metrics needs to be added and, in this sense, GG has a pretty straight-forward process that can be applied again just for the new parts and added to the previous created instance of the representation language and APIs.

Since a new framework demands a big developing effort, we decided to follow the Lean approach [15], which suggest starting building a Minimum Viable Product (MVP), which contains the essential features of the system. According to the Lean approach, the advantages of designing, implementing and evolving a MVP, instead of starting by a “complete/perfect” system design and implementation, is twofold: tests can be made early, enabling adjustments or significant changes before the developer is overcommitted with the system; and the design and development effort is more focused, and then more productive.

5.2 Overview of the GameGuts Architecture

The framework consists of the following key components:

- A gameplay representation language instantiated from the Game Ontology Project
- An API to store and access the recorded data (by the game and other servers)
- A visual query language to visually retrieve and present the recorded gameplay data.

A typical session starts with the player playing the game and his actions being recorded. The session is captured in the game client code then sent to the server. Every session that arrives at the server through the client API is stored in the gameplay database. The stored data can be accessed through the GG Server API or using the GG Visual Query Language. This section explains in details each component.

5.3 GameGuts Component: Gameplay Representation Language

In order to reduce the complexity on mapping and choosing detailed analytics for each part of the game and, therefore requiring less prior knowledge, GG provides a detailed systematic process. It is important to notice that although laborious, the process is simple and easy to understand. The process starts starts from the game entities and the Entity Manipulation branch of the GOP ontology, chooses the appropriate subset of terms from the branch that exist in the game and extends those terms to include the necessary properties and variables. Then, it is possible to instantiate a representation language of gameplay data from the GOP ontology that provides expressiveness to perform a variety of analysis in a computer-friendly format. To overcome the user-friendliness limitations of textual languages, there is also a process to create a Visual DSL, presented further along in this article, to improve the data retrieval and presentation.

The following steps instantiates the gameplay representation language:

1. Enumerate all the entities involved in the game, as well as the state variables and properties of each entity. Usually the game design document contains this information. For instance, in Pac-Man, the entities are the Pac-Man, the pellets he can eat along the way, the fruits which gives him powers and the ghosts that chase him - Inky, Blinky, Pinky and Clyde. Concerning the properties/states, the Pac-Man must have a position and a velocity, and whether he is under the power of a fruit or not is state variable.
2. Identify which parts of the Entity Manipulation branch is needed (which events occurs for each entity). The Entity Manipulation branch of the GOP is the part of the ontology, which describes all the actions performed by or in a game entity. This is the most crucial part of the process since it is the basis of the representation. It is important to consider all the manipulations in the GOP to make sure nothing is missing. The Pac-Man character, for instance, can move (ToMove) and collide with the other entities (ToCollide). We can represent a collision between Pac-Man and Blinky as ToCollide(PacMan, Blinky) or ToCollide(Blinky, PacMan). ToMove is also used for the Ghosts. Pellets and Fruits do not move, but are created (ToCreate) and also collides.
3. Adapt each Entity Manipulation element of the ontology providing the actual representation in terms of cardinality and valid parameters. This is necessary since GOP is usually superficial. For instance, the ToCollide class child of the Entity Manipulation part of the GOP does not provide time and point of collision. Then, for the ToCollide a few properties should be added to aid further analysis. A timestamp and the position of the collision is usually very useful. A collision record would look something like ToCollide(PacMan, Blinky, (27, 32), 5674) where (27, 32) would be the position of the collision (it can be in pixel or tiles depending on the game) and 5674 would be the timestamp in seconds or game loops (for replay purposes, game loops are generally a better idea). To record a move there are a few possibilities: using a velocity vector, using directions or using the previous and current position. In PacMan, for its simplicity, directions work well and a move to the left from Clyde would be like ToMove(Clyde, LEFT, 676).
4. Convert the Ontology in a DSL using Taira's [17] method.
4.1 The method requires the domain to be modeled as a class diagram. It's an extensive process but improves the separation of concerns. Figure 1, shows the class diagram for the PacMan game.
4.2 Transform the class diagram into a grammar in the Back-Naus Form [41]. The Pac-Man game BNF would have definitions like the following:

```
ToCreate ::= ToCreate(<entity>, <entity>, <position>,
<timestamp>)
ToCollide ::= ToCollide(<entity>, <entity>, <position>,
<timestamp>)
ToMove::=ToMove(<entity>,<direction>,<timestamp>
)
ToRemove ::= ToRemove(<entity>, <timestamp>)
<direction> ::= left | right | up | down
<position> :== (<number>, <number>,<number>)
<number> ::= <digit> | <number> <digit>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<timestamp> ::= <number>
EntityManipulation ::= ToCreate | ToMove | ToOwn |
ToShoot | ToCollide | ToRemove <timestamp>
<entity> ::= MAZE | PACMAN | BLINKY | PINK | …
```

5. Refactor the resulting DSL to use the JSON standard syntax to make the creation and parsing of the files easier. This step is optional but highly recommended. Using a new syntax will require the creation of a new parser, using a standard format makes things easier. Additional details are included to increase human readability of the generated logs. An excerpt of a Pac-Man session record would look like the following:

```
{ "entities":[
    {"id": 0, "type": "game.pacman.enemies.Inky"},
    {"id": 1, "type":"game.pacman.PacMan"},
    {"id": 2, "type":"game.pacman.Pellet"},
    {"id": 3, "type": "game.pacman.Pellet"},
    {"id": 4, "type": "game.pacman.Pellet"},
    {"id": 5, "type": "game.pacman.Maze"},
    {"id": 6, "type": "game.pacman.Fruit"}, ...],
  "EntityManipulations":{
    "ToCreate":{ "creator":5,
"created":1, "timestamp":403},
    "ToMove":{ "moving":1, "direction":"left",
"timestamp":1055},
    "ToMove":{ " moving ":1, "direction":"left",
"timestamp":1146},
    "ToCreate":{"creator":5, "created":0,
"timestamp":1507 }, …
    "ToCollide":{"entity":1,
"entity":6, "timestamp":7200},
    "ToMove":{ " moving ":1, "direction":"left",
"timestamp":7202},
    "ToMove":{ " moving ":1, "direction":"left",
"timestamp":7450},
    "ToCollide":{"entity": 0, "entity":
1, "timestamp":7580},
    "ToRemove":{"entity":0, "timestamp":7803}
  }}
```

5.4 GameGuts Component: API (GGAPI)

The API is broken into two parts: The client API, responsible for packaging and sending the session data to the server, and the server API that receives that data, stores it, and provides ways to access it.

5.4.1 Client API

Most of the state of the art previously presented in this article does not pay any attention to the client side. As stated before, this leaves a heavy burden in the developer that needs to spend effort in creating analytics code instead of the game code alone. To help developers in this matter GG provides a few guidelines on how to handle the game client code using the class diagram generated during while creating the gameplay representation language.

1. Every entity in the game has its manipulations already mapped. To maintain separation of concerns the class that represents the game entity and holds its properties (the model) is separated from the class that handles the manipulations (the change in the properties, the controller). For PacMan, the class architecture would work as in Figure 2.

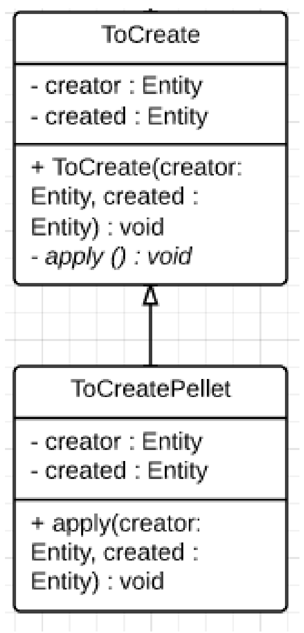

Figure 2: Class Diagram used in the client side of GG API

2. Use the Template Method design pattern [38] to abstract the record code from the game logic code. Using the Template Method helps to abstract the analytics code from the game logic code improving the code's separation of concerns and therefore its maintainability, reducing costs. A sample code for PacMan would look as follows.

```
public class ToCreate {
   public void ToCreate(Entity creator, Entity created, long timestamp) {
        // create the JSON entry, the entry is send to the server at the end of the session
        GG.getInstance().createJSONEntry(EntityManipulation.TO_CREATE, creator, created, timestamp);
        // call the subclass that actually implements this method
       this.apply(creator, created, timestamp);
  }
   protected abstract void apply(Entity creator, Entity create, long timestamp);
}
```

```
public class ToCreatePellet extends ToCreate {
  protected void apply(Entity creator, Entity create, long timestamp) {
      Vector3 nextPosition = ((Maze)creator).getNextPelletPosition();
      created.setPosition(nextPosition);
      ((Maze)creator).addEntity(created);
  }}
```

5.4.2 Server API

The Server side of the API is concerned about the storage of the logs that comes from the client and also to provide methods to retrieve that data. To improve performance, the GG server also provides triggers that are used to process the data when a log arrives from the client. The server side of GG, uses Google App Engine along with its NoSQL database [42]. Every log that arrives at the server is then processed by several triggers and stored in a various of formats according to different analysis needs. The raw data from the log is also stored and used when new analysis need the data in a format that is not available yet. Each of the analysis done in this article is explained in the corresponding section.

The server API also provides the following functions that can be used to access the data and are specially useful for the visual query languages.

1. View Log: Retrieves the whole log of a specific session. Example: Log of a given player in a given day and time.
2. Get Logs With: Retrieves the sessions logs that contains a specific list of Entity Manipulations. Example: All the logs where the player beats level 3. It's important to notice the Composite Action concept where you can represent a series of events that happens sequentially. It is very helpful to emulate universal and existential quantifiers.
3. Count: Counts the amount of times a specific Entity Manipulation appears in all logs. Example: How many times the player shoots with a pistol?
4. Count in: Counts the amount of times a specific Entity Manipulation happens in a given log. Example: For the log of a given player in a given date and time how many times he died?
5. Before: Lists all entity manipulations that happens before another specified. Example: How many times the boss of level five shoots his fireball before the player dies.
6. After: Lists all entity manipulations that happens after another specified. Example: How many times the player ship goes left after an iceberg spawns in the level?
7. Register Game Log: Used by the game client API to record a game session.
8. Register Visual Query: Used by the Visual Query Language to register a query (more of this in the next section).

5.5 GameGuts Component: Visual Query Language (GGVQL)

The GG Query Language (GGQL) is a language to query specific kinds of information from the database of session logs. The idea is to visually define gameplay scenarios or patterns and search the stored data for them [16]. GG Server API enables the creation of such tool and this section explains how to create the language, how it integrates with the server and how to use the language to retrieve information, all using PacMan as an example.

5.5.1 Creating the Visual Query Language

The Visual DSL consists of all the game elements, and the possible relationships between them (represented by the same actions mapped from the GOP into the UML class diagram). Every entity in the game is represented by its sprite in the game. This way, anyone can easily identify an entity in the query, and the entity manipulations are relationships between those entities. Ideally, to see how many times a PacMan collided with a Ghost, the query would look like Figure 3.

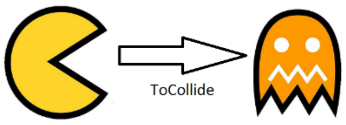

Figure 3: Visual representation of a query to see how many time a collision happens between PacMan and one of the ghosts.

Using Visual Studio DSL Tools solution [39] and its Minimal Language template creating such a language is simple. The language is created using the Classes and Relations partitions and the Diagram Elements partition is used to integrate the game sprites as elements of the language. Visual Studio itself can be used as a query editor. Each created query "compiles" into a program that registers the query in GG's server. GG Server then provides an interface to execute the query and see its results.

5.5.2 Integrating in the server

When a record arrives at the server, there is series of special triggers that adjust the data to be used by the visual query language. The following are used by the query languages:

1. Simple Counter trigger
Counts the occurrence of each entity manipulation. This means that every logs that arrives at the server is read and each entity manipulation (ToCollide, ToCreate, etc…) has its counter increased. This makes it simpler to understand how many times a collision happens per session, for instance.

2. Composite Action trigger
Creates a list of actions in the order they happened in the log. This trigger stores the session as a list of entity manipulations and provides faster retrieval for questions like: What is the accuracy of the player? Which can be done by querying how many times a ToCollide happens after a ToShoot event.

3. Game, Level and Time clustering triggers
At last, it is interesting to understand at which point of the level in a certain game the manipulations happen. This trigger separates each entry by their timestamp and level. It helps to identify things like: In Level 3, what is the part player's die the most?

5.5.3 Creating and using queries

As states before, using Visual Studio DSL Tools is pretty simple. The user creates a query and defines a name for it (the server lists the registered queries by name). Using the editor from Visual Studio all the user needs to do is drag and drop the entities and manipulations involved in his query.
The player then compiles the query and executes it, registering the query in the GG server. All the queries created are stored. The server creates a trigger, if necessary, to facilitate the query. It also includes the query in a list of queries created. The user then access the GameGuts server and choose the query to be executed. GG shows the results. The following section shows how the language was created for Imuno and how it was useful to find design flaws.

6. Case Study: OjE

To be validated, GG was used in a gaming and learning platform called OjE, which contains various games made in Flash [40] and is used in public schools in Brazil by over than 100,000 students. Each game in the platform was changed to include the GG client API and, consequently, record its gameplay sessions. Over half a million sessions, from all the games, were recorded using GG. This section shows how it was used to acquire, represent and analyze data for one of the OjE games: Imuno, a River Raid clone in which the player is a medical nanobot that needs to eradicate anything harmful to the human body. Figure 4 illustrates the game. At the end of the section improvements over the QA process applied to all the games are also discussed.
Besides the changes performed to include GG client API in the games, another change was done to create a version of the game that used the recorded log instead of the player's input to control the main character. This helped to ensure the gameplay representation language was expressive enough to hold all the information in the session therefore enabling designers to watch a replay of any record. The ability to watch a replay was essential for a handful of analysis. First, it helped game designers to analyze the session of dozens of players. Second, when a bug was found, it was easy to reproduce and, therefore, reducing a large amount of testing and bug fixing effort.

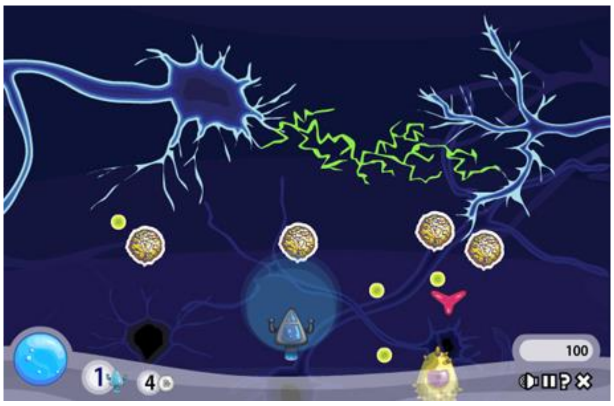

Figure 4: Imuno

6.1 Imuno: Game Design Analysis

Imuno is a action shoot 'Em Up game. It is of special interest to designers to identify the most difficult the parts of the levels that are too easy or too difficult. This information allows designers to understand the pace of a created level and adjust it to the desired player experience. A few queries were created in Imuno to perform this pace analysis. First, parts of the level (that we call scenario) where the player died were identified by searching for the tag ToRemove involving the player. This gave the designers a good understanding of frustrating scenarios. Then, the analysis tried to identify the scenarios that were too easy, i.e., the parts of the levels where the player did a lot of shooting and without dying. Using queries like that shown in Figure 5, , the designers were able to identify that the end of the first level (more precisely in the boss battle) was one of those "easy scenarios".

Another scenario of interest for game designers, where the player health reached very low numbers and could come back to continue playing were called "near-death experiences". It was desired to have as many of these scenarios as possible since it provides a neither too hard or too easy challenge. Being able to watch replays of those scenarios helped game designers to recreate them in several levels.

Looking for easy parts was also useful to to identify design flaws. On the most critical one was found in one of the game's bosses. Players could stand in a certain part of the screen without ever being hit. The boss spawned mobs periodically and the player would keep killing them from the safe position without ever hitting the boss too. This way, the players that found the glitch, could stay forever killing the helper mobs and, potentially, achieving infinite score. Although a careful analysis of the level could lead to the same conclusion, the game was played by OjE players for several months without the designers ever understanding why it happened. As soon as GG was implemented, with the ability to search the logs and watch a replay of the match the issue was understood in a few minutes.

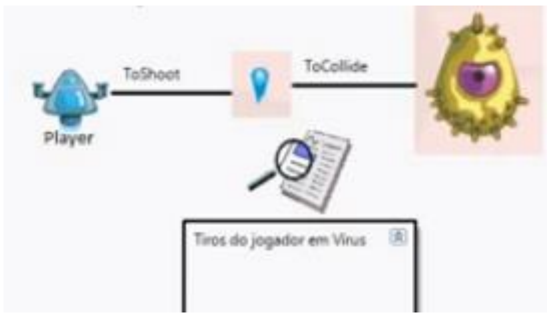

Figure 5: The query created using the Visual Description Language Tools in Visual Studio 2010

6.2 Imuno: Security Rules

One of the biggest issues in online games are cheating players. Using cheats to improve the score is not only fair but also provokes mass evasion from regular players that finds it impossible to beat a cheating player. To prevent that, each log that arrives at the server is analyzed in search for bugs. Attaching a trigger in the server reads each event in the log transform it to facts in a Drools rule database [32]. Then, rules are ran against the data to identify whenever someone is suspected of performing one of the most common cheats. For instance, some of the rules inspect all the collisions occurred between an enemy and a bomb or a bullet. Only in these situations should the score change. After running the whole session, the score is checked against the score from the client message and indicates cheating whether they differ.

Using GG, a cheat analyst needed about four to eight hours to code rules to catch a cheat type, leading to a total of eighty hours for all the rules of Imuno. The time spent in coding rules is by far compensate by the fact that large logs can be analyzed fastly. Pure manual analysis (without GG) would be nearly impossible. In fact, taking the time required by the same analyst to examine very small logs looking for the same kind of information, the analysis of all logs from OjE would require approximately one million hours (two hours of analysis per log).

```
rule "Bullet-Enemy Collision"
  when
     $e : Enemy( isDead == false );
     $p : Player();
     $b : Bullet(createdBy == $p.ID);
     $c : ToCollide (
        (entityA == $e.ID || entityB == $e.ID)
        && (entityA == $b.ID || entityB == $b.ID) )
  then
     $e.setHealth($e.getHealth() - $p.getDamage());
     retract($b);
     update($e);
     System.out.println( "Enemy lost some health: " +
$e.getID());
end
rule "An enemy has died, update player's score"
  when
     $e : Enemy (health <= 0, isDead == false);
     $p : Player();
  then
     $p.setScore($p.getScore() + 200);
     $e.setIsDead(true);
```

```
    update($e);
    System.out.println( "Player got some score from: " +
$e.getID() );
end
```

6.1 OjE: Quality Assurance Automation

Being able to replay sessions also helped a lot in the testing process for OjE. As explained in the beginning of the session, the bugs were easily reproduced with the change to use logs as input. This adjustment also made it possible to automate some conformance tests, regression tests (where all the bugs ever found is tested again to see if a change in the code broke the game again and the “bug is back”) and also exploratory tests.

Conformance tests were easier since the tester only needed to create the scenario and play the game once. Any bug found could be easily reproduced and the test could be made several times using the replay system without any additional human effort. The exploratory test for most of the game was made by creating a random player. This way, several scenarios not foreseen by the game testers could be tested. All the games were tested using this approach, which is not new but was greatly improved and facilitated by the use of GG.

7. Conclusion

The current existing frameworks are very good in providing game analytics in general, with a special attention to the ARM coarse-grained metrics. However, when dealing with fine-grained data, the frameworks do not offer enough support or guidance to decide which data to capture, to write the code to capture it, to choose the best representation of it and to allow an adequate retrieval and presentation of it.

In this paper we have presented GameGuts (GG) as an original attempt to provide better assistance to game developers to work with fine-grained data through a (i) step-by-step process on how to define, represent and code fine-grained data, (ii) an API to retrieve data and, finally, (iii) a visual DSL to present the data in a human-friendly way.

GG was successfully used in OjE, a framework composed by six casual game played by over 100.000 users, especially in the Imuno game. This proved the efficiency and usefulness of the framework for OjE case. However, in order to get a better understanding on GG limitations, as well as to improve it, it is necessary to use it in games created by other developers. Not only to see GG limitations but also to test some of the hypothesis (e.g. is GOP simple for developers or it still requires too much prior knowledge, does it helps the separation of concerns). Therefore, we intend to adapt GG to be compatible with other existing frameworks, such as Google Analytics and Flurry, in order to allow the developers that already make use of these frameworks to adopt GG more easily and, after that, be able to interview the users of GG in order to understand its strengths and weaknessess. This is just a matter of implementation, since conceptually GG can couple with some of the existing tools for coarse-grained data analysis as a complementary approach.

Acknowledgements

The author would like to thank Joy Street S/A, maker of Olimpiadas de Jogos e Educação (OjE) and owner of Imuno for allowing this work and providing the game’s source code and access to programmers and designers involved in the production. And our supporters and sponsors: Cin – UFPE, CAPPES and Manifesto Game Studio LTDA.

References

1. Glasser, The Smurfs & Co Top This Week`s List of Fastest-Growing Games by DAU [Online] Available: http://bit.ly/1jbQWnA [Accessed: May 12, 2014]
2. Zagel, J.P., et al. Towards an ontological language for game analysis. In: DiGRA 2005 Conference: Changing Views – Worlds in Play 2005
3. J. ZAGAL. Game Ontology Project http://www.gameontology.com/index.php/Main_Page
4. J. T. C. Chan, W. Y. F. Yuen. Digital game ontology: Semantic web approach on enhancing game studies. In: 9th International Conference on Computer-Aided Industrial Design and Conceptual Design, 2008. CAID/CD 2008.
5. El-Nasr, S.; Drachen, A.; Canossa, A. (Eds.). “Game Analytics”, 1st ed. New York, Sprint, 2013.
6. W. Memphan, “Semantically enhanced games for the Web”, In: Proceedings of the Web-Sci'09: Society On-Line, Athens, Greece, 2009
7. Flurry. [Online]. Available: http://www.flurry.com/ [Accessed: November 30, 2013].
8. J. Yan, B. Randell, An Investigation of Cheating in Online Games. In: IEEE Security & Privacy, vol. 7, no. 3, May/June 2009.
9. W. Stallings. Cryptography and Network Security: Principles and Practice. 4. ed. New Jersey: Prentice Hall, 2005.
10. Hürsch W. L. and Lopes C. V. Separation of Concerns. Northeastern University, Boston, USA. Technical report NU-CCS-95-03, February 1995.
11. J. A. Whittaker What Is Software Testing? And Why Is It So Hard? IEEE Software, v.17, n.1, p.70-79, Jan./Feb 2005.
12. Miryung Kim, Lawrence Bergman, Tessa Lau, David Notkin. An Ethnographic Study of Copy and Paste Programming Practices in OOPL. In Proceedings of 3rd International ACM-IEEE Symposium on Empirical Software Engineering (ISESE’04), pp. 83- 92, Redondo Beach, CA, USA, August 2004.
13. Thompson, C. Halo 3: How Microsoft Labs invented a new science of play. Wired. August 8, 2007. http://www.wired.com/gaming/virtualworlds/magazine/15-09/ff˜halo. 2007
14. Olimpiadas de Jogoes e Educação (OJE) [Online] Available: http://www7.educacao.pe.gov.br/oje [Accessed: July 12, 2013].
15. Ries, E. "The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses", Crown Publishing 2011
16. W. B. FURTADO. SharpLudus: Improving Game Development Experience through Software

Factories and Domain-Specific Languages. MSc thesis, Federal University of Pernambuco. 2006
17. R. Tairas, M. Mernik, J. Gray, Using ontologies in the domain analysis of domain-specific languages. In: Proceedings of the 1st International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering 2008. CEUR Workshop Proceedings., CEUR-WS.org, vol. 395 2008
18. Hazewinkel, M, "Predicate calculus", Encyclopedia of Mathematics, Springer, 2001
19. Richard E.; "Dynamic programming". Princeton University Press. 1957
20. O. Sotamaa, T. Karppi. Games as Services Final Report. TRIM Research Report 2010.
21. M. J. Nelson Game Metrics Without Players: Strategies for Understanding Game Artifacts. AAAI Technical Report WS-11-19 2009
22. Drools [Online] Available: http://www.jboss.org/drools/ [Accessed: July 26, 2013]
23. P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, E. Postma, Adaptive game ai with dynamic scripting. Machine Learning, 63 (3), 217–248. 2006
24. T. Schaul Evolving a compact concept-based Sokoban solver. Master's thesis, EcolePolytechnique Federale de Lausanne. 2005
25. P. Spronck, I. Sprinkhuizen-Kuyper, E. Postma, Online adaptation of computer game opponent AI. In Proceedings of the 15th Belgium-Netherlands Conference onArtificial Intelligence, pp. 291–298. 2003
26. N. Sorenson and P. Pasquier "The evolution of fun: Automatic level design through challenge modeling", Proc. 1st Int. Conf. Comput. Creativity, pp.258 -267 2010
27. K. Chiu, K. Chan, Using data mining for dynamic level design in games. Proceedings of the 17th international conference on Foundations of Intelligent Systems 2008
28. Pedersen, J. Togelius, and G.N. Yannakakis, "Modeling Player Experience in Super Mario Bros," Proc. IEEE Symp. Computational Intelligence and Games. pp. 132-139, Sept. 2009.
29. J. Williams, Acquisition, Retention, Monetization: Understanding & Optimizing player behavior on any plataform, http://casualconnect.org/lectures/monetization/acquisition-retention-monetization-josh-williams
30. Crash Reporter [Online] Available: http://en.wikipedia.org/wiki/Crash_reporter [Accessed: July 26, 2013]
31. D.L. McGuinness and F. van Harmelen eds. "OWL Web Ontology Language Overview," World Wide Web Consortium (W3C) recommendation, Feb. 2004, www.w3.org/TR/owl-features.
32. Jess, The Rule Engine for the Java Platform [Online] Available: http://herzberg.ca.sandia.gov/ [Acessed: July 26, 2013]
33. Music Ontology Specification [Online] Available: http://musicontology.com/ [Accessed: July 26, 2013]
34. M. Smith , M. J. Nelson and M. Mateas "Ludocore: A logical game engine for modelling videogames", Proc. IEEE Conf. Comput. Intell. Games, 2010
35. J. Stenros, O. Sotamaa. Commoditization of Helping Players Play: Rise of the Service Paradigm. In Proceedings of DiGRA 2009: Breaking New Ground: Innovation in Games, Play, Practice and Theory. 2009
36. Shacknews.com, EA continues to add servers to alleviate SimCity crush, [Online] http://www.shacknews.com/article/78127/ea-continues-to-add-servers-to-alleviate-simcity-crush [Accessed: July 26, 2013]
37. Arstechnica.com, Blizzard fixes Diablo III gold duplication bug, but the damage may be done, http://arstechnica.com/gaming/2013/05/blizzard-fixes-diablo-iii-gold-duplication-bug-but-the-damage-may-be-done/[Accessed: July 26, 2013]
38. E. Gamma, et al. Behavioral Patterns *"Design Patterns: Elements of Reusable Object-Oriented Software*" Addison-Weles 1994 pp 325
39. Overview of Domain-Specific Language Tools [Online], http://msdn.microsoft.com/en-us/library/bb126327.aspx [Accessed: July 23, 2014]
40. Flash Developer Center [Online] http://www.adobe.com/devnet/flash.html [Accessed: July 23, 2014]
41. ISO/IEC, EXTENDED BNF 1996 [Online] www.dataip.co.uk/Reference/EBNF.php [Accessed: July 23, 2014]
42. Rick Cattell, 2010. "Scalable SQL and NoSQL data stores," ACM SIGMOD Record, volume 39, number 4, pp. 12–27
43. Marketing Tech, App Annie report reveals freemium rules for app revenue [Online], http://www.marketingtechnews.net/news/2014/mar/28/app-annie-report-reveals-freemium-rules-app-revenue/ [Acessed: September 7th, 2014].