Steam Games and Reviews

Ângelo Teixeira, Duarte Frazão, Mariana Aguiar and Pedro Costa
Faculty of Engineering, University of Porto
{up201606516, up2016005658, up201605904, up201605339}@fe.up.pt

***Abstract*—Through the evolution of the internet, the quantity of information has grown to an unimaginable level, reinforcing the importance for developers to create mechanisms capable of connecting available information. Such is the case for searching video-games, where big companies control the most relevant online stores not utilizing any information outside of their domain, leaving behind much possible information on the table. In this paper, we present a broader search system than the one found on Steam. To do so, we use two datasets available on Kaggle containing information about all the current games in Steam and the reviews of a subset of them. This information is then refined and enriched with information from Wikipedia and conceptually described. Exploratory data analysis is conducted to understand possible patterns, list possible retrieval tasks and prepare it for the retrieval phase. We utilized the Solr search platform to implement the information retrieval system, defining the collection and the respective documents, while describing the indexing process. A set of information needs is created from the previously defined retrieval tasks to explore the selected tool's capabilities and evaluate the retrieval system. Results are satisfactory and provide further insights into the importance of specific fields; specifically, the reviews are essential in obtaining correct results when the information need portrays a general negative aspect of games. Finally, in the Semantic Web domain, we explore existing ontologies applicable to our domain and, with the Protege tool, define and populate our own ontology — answering the remaining retrieval tasks through SPARQL queries. The obtained results were satisfactory but limited given the tool's constraint on the dataset size.**

***Index Terms*—Data Extraction, Data Refinement, Data Analysis, Search Engine, Steam, Video games**

I. Introduction

The current landscape of gaming search systems is still lagging in the information that it can offer users, using only what is already available inside the stores. At the moment, Steam [11], the most popular store for desktop games, only allows users to search for games' titles with a limited filtering and ordering section. The motivation for tackling this problem is to present a broader search system for users looking to buy a new game, with more and better text search capabilities that provide relevant information about the games and their reviews. The paper is divided into two three parts. The first details the process of data retrieval and preparation. Initially, pre-collected datasets with information relevant to the problem at hand were selected and analyzed. The data then went through various refinement tasks and was enriched with data extracted from Wikidata [19], and Wikipedia [20] APIs. After that, the data was analyzed and characterized to understand its values and distributions better, resulting in a conceptual model. Finally, possible retrieval tasks are presented to identify the possibilities the final product can have. The second part presents and analyzes the implementation of the information retrieval system for our data focused on ad hoc search. We start by defining the three different versions of the system that will be used for evaluation and comparison, along with a justification for the tool used to create these systems. From the previously defined possible retrieval tasks, we create six Information Needs (IN) to evaluate the systems. Finally, we present the results and a discussion for each IN followed by an analysis of the previously selected tool. The third and last part explores the dataset from a Semantic Web perspective, creating an ontology for our domain. We start by describing the Semantic Web's origins and motivation and existing ontologies applicable to our domain. Then, we describe the process behind our ontology's creation and population in the Protégé [16] tool and answer the remaining retrieval tasks with SPARQL queries. Finally, we evaluate both the queries results and the tool itself, followed by a comparison between the Information Retrieval and Semantic Web paradigms.

II. Data Extraction

To select the most appropriate datasets for the chosen domain and the problem at hand, various open data platforms were consulted. With the goal of finding a dataset rich in structural data and another one rich in textual data, Kaggle [5] and Zenodo [22] proved to be the best solution. Kaggle is a platform for data scientists and machine learning practitioners to publish data sets and explore and build models. Zenodo is a plataform that allows researchers to deposit data sets, research software and other research digital artifacts.

For the structural data, the Steam Store Games dataset [3], found on Kaggle, was selected. The dataset provides information about various aspects of the games on the Steam Store, such as its developer, categories, genre, price and the estimated number of owners. The data was pre-collected from the Steam Store [11], owned by Valve, and SteamSpy [12] APIs, owned by Sergey Galyonkin, and only represents a subset of the whole domain, more specifically, prior to May 2019. It is a dataset composed of 6 *csv* files, of around 50 MB and with 27k rows per file.

For the textual data, the Steam Review dataset [7], found on Zenodo, was selected. The dataset provides reviews of some selected Steam games with a review sentiment and the number of users who marked the review as helpful. The data was pre-collected from the Steam reviews portion [10] of Steam store and only represents a subset of the whole domain. It is a dataset composed of 1 *csv* file, of around 2 GB and with 6.4M rows, resulting in **33%** of games having at least one review.

In order to add another data source to the whole set of collected data, detailed description about some of the games' publishers and developers was gathered using the Wikidata SPARQL API [19] and the Wikipedia API [20]. This process was considered an enrichment task and is described in detail in Section III-B.

III. Dataset Preparation

In order to make sure the collected data is ready for an information retrieval phase and as accurate and correct as possible, some refinement and enrichment tasks were performed on both datasets.

Prior to starting the data preparation, the selected datasets were studied in terms of the information they offer, so that, only, the attributes relevant for the possible search tasks and domain would be selected. This exercise was especially important for the Steam Store dataset given that it was, initially, consisted of 6 different files and resulted in a single file with attributes aggregated from 3 files.

An overview of the workflow pipeline executed to prepare the datasets can be consulted in Figure 1.

A. Refinement

The tasks performed for the data refinement of the chosen datasets involve the selection of attributes, transformation of attributes' data types, transformation of attribute values and are described in more detail below.

The data refinement was accomplished using OpenRefine [15], a free, open source tool for cleaning and transforming data.

1) Attribute Selection: As mentioned previously, the most relevant attributes were selected to create the final dataset and OpenRefine [15] was used to delete the extras and join others from the different files. The Steam store file was composed, initially, of 18 attributes, from which only 2 were deleted, *steam_tags* and *achievements*. The first due to containing, in its majority, duplicated information present in the *categories* and *genres* attributes; and the second because it was considered not relevant to the domain. Additionally, 2 attributes, *website* and *detailed_description* were joined from other files.

2) Attribute type transformation: To normalize the formats of the attributes of the same type, the following transformations were performed. For the binary attributes (0 and 1), such as *english*, the values were mapped to boolean values, True and False. Finally, the *owners* attribute, originally in a bucket interval format was transformed into numeric values by considering the maximum limit of the interval.

3) Attribute cleaning: Upon further inspection of the *publisher*, *developer* and *name* attributes, characters and expressions considered not necessary were discovered and removed. These expressions could also make the search for the attribute values in other data sources impossible. Some examples are characters like ™ and © and expressions like "Lda.", "LLC" and ".inc". Whenever possible, the capitalization of the previously mentioned attributes was normalized between objects with the same values.

B. Enrichment

In order to better relate publishers and developers – which we will refer to as organizations – and to present more information about them, besides the title, we used both the Wikipedia and Wikidata API to obtain a general description of the companies. The querying was done through two *Node.js* packages [21] [18].

The process is divided into two phases and each phase has two steps. Each game entry on the *csv* file contains the name of both organizations. However, we noticed that the name is not always correct, or sufficiently accurate to get a match from Wikipedia. The name of the game is, usually, more reliable.

The first phase begins with the name of the game. Given the previous issue, we start by querying, with SPARQL, the Wikidata API with the name of the game to obtain the correct names of the organizations (first step). After that, we use those names to directly query Wikipedia and get a text extract (second step).

Nonetheless, we also noticed that some games do not have an entry on the Wikidata knowledge base but their organizations do. In the second phase, for these missing matches we use the organizations names present on the *csv* and executed the same steps as mentioned above, only instead of querying games, we query publishers or video-game developers.

Considering that the name obtained by the API can be slightly different that the ones already present on the file, we update each row accordingly, thus maintaining the relation. In the end, **15%** of games have a publisher and developer with a description. The output of this process is stored in a *csv* file.

IV. Conceptual Model

After all the data preparation tasks, especially, the attribute selection in the refinement phase and the enrichment of the original datasets with information from Wikipedia and Wikidata, the conceptual model of the data is represented in Figure 2.

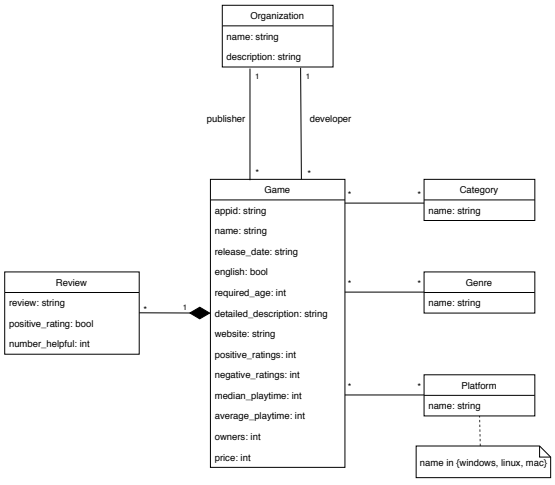

Fig. 2: Conceptual Model.

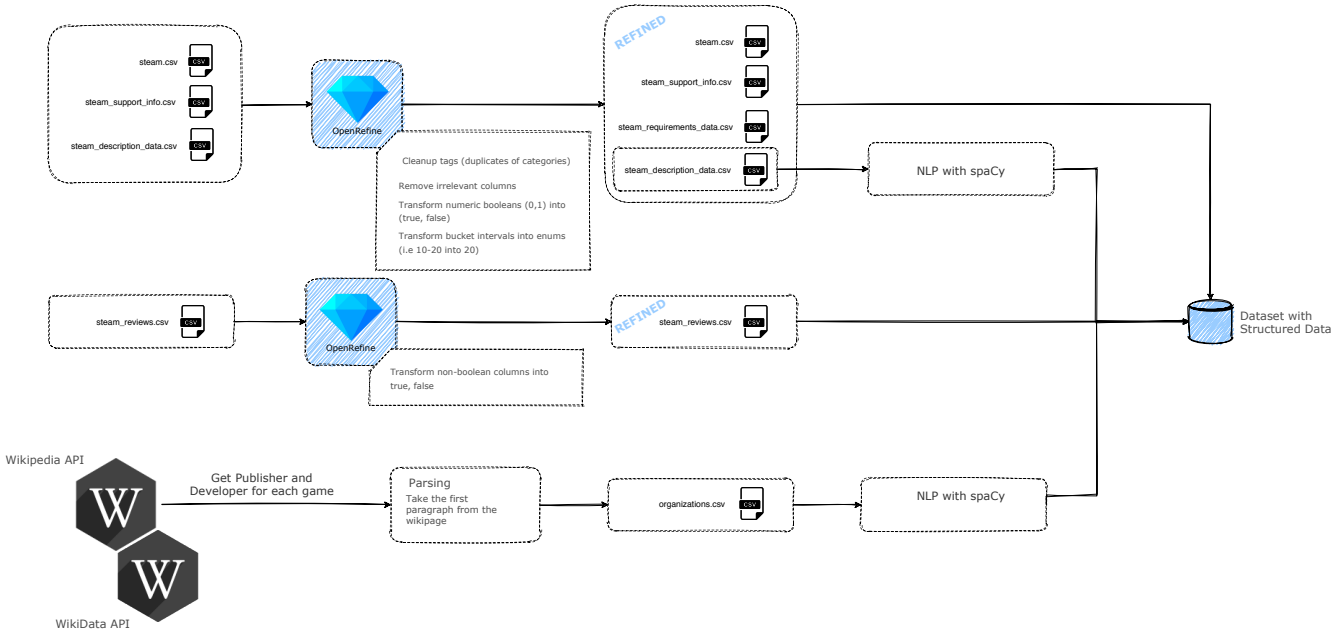

Fig. 1: Workflow Pipeline.

There are 3 main classes: *Game*, *Review* and *Organization*. The *Game* class has attributes that describe the game objects and attributes that will be useful for the users, in a later phase, to understand if the game is in general well rated and popular. This class also has associations with other classes, *Category*, *Genre* and *Platform*, that help to characterize the game. The *Review* represents a review of a certain game and the class *Organization* represents the entities that publish and develop the games.

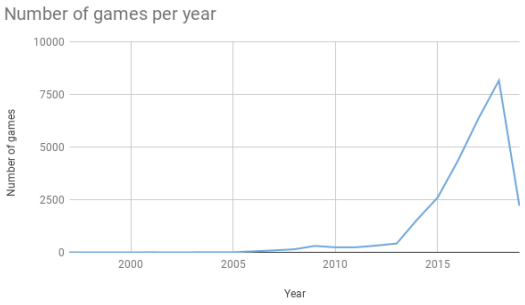

Fig. 3: Number of games per year.

V. Data Characterization

In this section we explore the distribution and characterization of the collected data.

A. Games distribution

As expected by the boom in the gaming industry from the past fifteen years most of the games we have in our dataset are from that time frame, as we can see from Figure 3. We're also able to analyse that we have games dating back to as much as the year 1997, with a significant increase starting at the year 2005, which coincides with the year Steam started negotiating contracts with other publishers to release their games on the Steam store.

In Figure 4 we're able to see the genre distribution, Indie being the most predominant, which could make the addition of the publisher and developer information even more relevant for our search system.

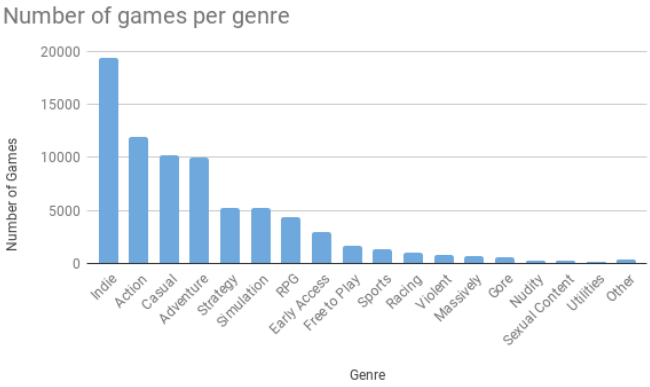

Fig. 4: Number of games per genre.

B. Text Analysis

There are three main textual fields in the system: organizations descriptions, reviews and game descriptions. The length of each one was analysed and the results can be seen in Table I.

Additionally, we used spaCy [8] in order to analyse the entities contained in each fields. We found that both in the reviews and in the game descriptions, there wasn't a relevant (and consistent) number of entities that would be useful for us (*e.g.* entities that could link different games or reviews). However, with the descriptions of organizations, there was a significant mean number of entities extracted – mostly people and other organizations – that can prove to be useful in connecting different games. The results can be seen in Figure 5.

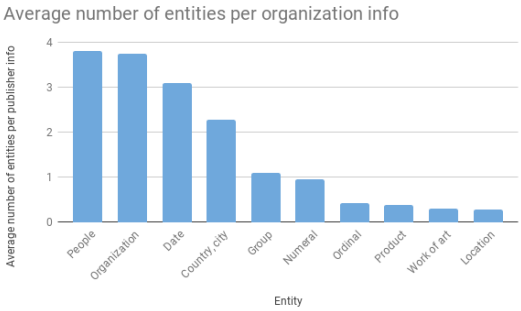

Fig. 5: Average number of entities per organization description.

TABLE I: Text attributes characterization.

	Organization Descriptions	Reviews	Game Descriptions
Total number	1,011	6,417,105	27,334
Average length	729	304	1634
Length p15[1]	308	30	837
Length p50	526	104	1,303
Length p75	983	310	2,026
Length p95	1,975	1,237	3,907

VI. Retrieval Tasks

The retrieval tasks possible with the collected data are focused on 2 documents: Games and Organizations. Some of the established retrieval tasks are:

- Search for games
- Search for organizations (games' publishers and developers)
- Search for reviews
- Search for games' categories
- Search for games' genres
- Filter games by year, publisher, developer, price, number of owners, platform, ...
- Filter reviews by sentiment and number of helpful votes
- Order games by price, playtime, positive and negative ratings, number of reviews, release date and number of owners
- Order reviews by number of helpful votes
- Top reviews of a game
- Related games based on categories/genres/publisher/developer

Currently, Steam offers its users the ability to search for games and to filter them by tag, type, number of players, features, platforms, language and price. In addition to this, with the data collected, it will be possible to search for organizations and filter the games per publisher, developer, number of owners, required age, playtime and positive and negative ratings.

VII. Information Retrieval

An Information Retrieval System primary goal is to retrieve the documents of a collection that are relevant to a user information need while retrieving as few non-relevant documents as possible. This study will be focused on the most common retrieval task, ad hoc search. Ad hoc search is a task where the user specifies their information needs through a textual query that is performed on all documents of the collection and where the obtained results are ordered according to how relevant they are.

In order to study and to evaluate the results of possible retrieval tasks on the Steam Games domain, specified previously, we will use three different systems: (1) a baseline system where basic matching on key fields is performed, (2) a system where the main textual fields were indexed with the use of filters (presented in Section X) and (3) a system with weights where relevant fields are boosted in the retrieval process.

A set of information needs, defined in Section XI, will be translated to a textual query and tested in each system. Due to the high number of documents, each query will be evaluated on their Precision @ 10 (P@10) and Average Precision (AvP) values. P@10 measures the precision, the fraction of retrieved documents that are relevant, at fixed low levels of retrieved results and AvP is the average of the precision values obtained for the set of the top k documents existing after each relevant document is retrieved. To compare the global performance of the three systems we will use the Mean Average Precision (MAP), obtained by averaging the AvP values of all the information systems.

VIII. Information Retrieval Tool Selection

The two main tools considered to be used for the information retrieval tasks were Apache Solr [13] and Elasticsearch [14]. Both tools are open-source, built on top of Lucene nodes and offer a variety of features, such as, distributed full-text search, near real-time indexing, high availability and support for NoSQL data. Solr is an older tool and so, it is a more mature product with a broad user community. It offers features like faceting, integration with big data tools, support for rich-text documents and complex search queries that are

[1]pX is shorthand for Xth percentile

unavailable in Elasticsearch. Even though Elasticsearch is more recent than Solr, it is the most popular search engine since 2016 [4]. It only supports JSON, has a powerful indexing and searching functionalities and offers more scalability features than Solr.

However, the main reason for our final choice was the use cases of each tool. While, Elasticsearch is focused on scaling, data analytics and processing time series data, in order to, extract meaningful insights and patterns; Solr is best suited for search applications that use significant amounts of static data. The problem at hand, described in the previous sections, falls in Solr's use case: a search application on Steam games data to perform advanced information retrieval tasks.

One disadvantage of Solr, that we came to discover while using the search tool, is its poor documentation and lack of good technical examples and tutorials. Elasticsearch, on the other hand, is known for its well organized and high quality documentation.

IX. Collections and Documents

There are two different types of documents in our system: games that represent the best-selling Steam games and correspond to about 27,000 of the documents and reviews of the best-selling Steam games that are considered children of the game documents with about 6,400,000 objects.

Initially, all the data was in *CSV* files and while Solr allows indexing of documents from files in *csv* format, it does not support that these documents are processed as child documents. In order to index the reviews as nested documents the games and reviews data were converted to the *JSON* format and aggregated in a single file with an array of *JSON* objects.

With a single *JSON* file containing the two types of documents, we were able to index all the documents in a single collection where all the information needs will be queried upon. The association between the games and reviews documents is assured by Solr's nested document feature and can the queried using the `child of` and `parent which` keywords.

X. Indexing Process

Before starting the indexing process, we studied which fields from each document should be indexed. We decided to only index the fields necessary to fulfill the previously defined information needs, by serving as a query parameter. The final indexed fields of the game documents are: name, detailed_description, categories and genres. The indexed field of the review documents is only the review text. The final schema can be consulted in Table II. The documents were indexed using Solr's Post tool.

The indexed and stored fields with numeric values were defined using the default Solr field type *pint*, most of the textual fields using the string or strings (when multi valued) field type and the dates with Solr's *pdate* field type. The most important textual fields, such as the games' name and description and the reviews' text were subjected to a analyzer pipeline, that includes a tokenizer and optional filters for further processing each generated token. To achieve this we created three custom field types for each category of textual attribute: names, tags and small paragraphs.

TABLE II: Schema fields.

Field	**Type**	**Indexed**
average_playtime	plong	No
categories	tag_text	Yes
detailed_description	custom_text	Yes
developer	name_text	No
genres	tag_text	Yes
is_english	boolean	No
median_playtime	pfloat	No
name	name_text	Yes
negative_ratings	pint	No
owners	pint	No
platforms	strings	No
positive_ratings	pint	No
price	pfloat	No
publisher	name_text	No
release_date	pdate	No
required_age	pint	No
website	string	No
review	custom_text	Yes
number_helpful	pint	No
sentiment	pint	No

The three custom field types, *name_text*, *tag_text* and *custom_text*, use Solr's standard tokenizer but differ in the filters that each one applies. The field type *name_text* used in the game documents' categories and genres fields applies a lower case filter that converts all tokens to lower case. This filter helps combat possible relevant documents not matching due to case mismatch. The field type *name_text* intended to be used in the games' names, in addition to the lower case filter it also applies a synonym filter. The configured synonyms are mostly common acronyms for popular games, that are usually searched for instead of the full game title. The *custom_text* field type defined for longer blocks of text, such as the games' description and the reviews' text also applies the lower case and synonyms filter. On top of them, *custom_text* applies a filter to remove English stop words, one to remove possessives and a stemming filter to reduce the vocabulary size and increase matching. A summary of the custom field types can be seen in TableIII.

TABLE III: Custom Field Types.

Field Types	Filter	Index	Query
name_text	LowerCaseFilterFactory	X	X
	SynonymGraphFilterFactory		X
tag_text	LowerCaseFilterFactory	X	X
custom_text	StopFilterFactory	X	X
	LowerCaseFilterFactory	X	X
	EnglishPossessiveFilterFactory	X	X
	PorterStemFilterFactory	X	X

Although we couldn't find any list of games abbreviations we created a script that followed the general pattern of abbreviation for games that returned the valid acronyms, *i.e.* extract the first letter of each word in the game title, excluding games from the same game series. As we'll explore in Section XI this synonym list added the possibility to search for games using their abbreviations (e.g., cs - Counter Strike, tf2 - Team Fortress 2). A preview of the file can be seen in Listing 1 .

Listing 1: Synonyms file.

```
cs => counter strike, cave story, castle
    story
tfc => team fortress classic, train
    frontier classic
...
```

XI. Retrieval Process

To evaluate the different systems, we defined 6 different information needs (IN) based on the possible retrieval tasks, previously defined in Section VI. For each information need, we provide a simple description and the relevance judgment to decide if a document is relevant or not. To evaluate each system performance for each information need, we analyzed the top 10 results, their relevance and calculated the P@10 and AvP, as mentioned previously.

The query fields used were the indexed ones, *i.e.* detailed_description, review, categories and genres. The name field was only used in the information need present in Section XI-F.

In the weighted system, we followed an ad hoc approach to determine the weights of each field. The result can be found on Table IV.

TABLE IV: System 3 - Weights Distribution.

Field	Weight
detailed_description	1.5
review	0.5
categories	0.1
genres	0.1

As seen in Section IX, the number of reviews is two orders of magnitude higher than the number of games. This difference is even more noticeable when querying Solr without weights, the top results are usually only reviews documents. Thus, we decided to reduce the importance of reviews in the weighted system to assess the importance of this type of documents in the defined INs.

A. Family Games

Information Need: Games suited for families that are multiplayer, have a low required age and do not have violent or sexual content.

Relevance Judgment: Here we intend to retrieve games that are suitable to play with your family. They should be rated for ages < 12, have a multiplayer mode, and the description should mention things like "family", "fun for all" or "couch play". We can find this information on the game categories but also on reviews from players that let the others know that they had a great time (or not) with their kids while playing the game.

Query: `(family OR "fun for all" OR kid) AND multiplayer`

Conclusion: The first two systems had a better performance on average than the third, as it can be seen in Table V. System 2 is slightly better and has the best R-Precision value (83%), meaning that its relevant results appear sooner than in the other systems. This is due to the fact that some games might mention "kidding" or words like "familiar" which can deem the document relevant to this Information Need without actually mentioning the exact query terms. In the third system, since we value the games' descriptions way more, it's "ignoring" some of the reviews that more relevant games have, that show up on system 1, making this last system worse in terms of precision and also recall.

TABLE V: Family Games Information Need Results.

Rank	System 1	System 2	System 3
1	N	R	N
2	R	N	R
3	R	R	N
4	R	R	N
5	R	R	R
6	N	R	R
7	N	N	R
8	R	N	N
9	N	N	R
10	R	R	N
P@10	0.60	0.60	0.5
AvP	0.657	0.775	0.505

B. Online games with server problems

Relevance Judgment: Here we intend to retrieve games that are suitable to play online, but have lag or server issues. They should be online games and mention connection or lag issues in the reviews to be considered relevant. We cannot look at the games description, since the developers won't mention it there.

Query: `(lag OR "server down") AND online`

Conclusion: The first two systems had a way better performance than the third, as it can be seen in Table VI. Due to the fact that System 3 has a higher weight on the games' descriptions, and when they have "lag" references is to mention that they are "lag-free", it will often consider games as relevant when they are not, which reflects on its precision of 0.

TABLE VI: Online games with server problems Information Need Results.

Rank	System 1	System 2	System 3
1	R	R	N
2	R	R	N
3	R	N	N
4	R	R	N
5	R	N	N
6	N	N	N
7	R	N	N
8	R	R	N
9	R	R	N
10	N	N	N
P@10	0.80	0.5	0.00
AvP	0.953	0.76	0.000

TABLE VII: Free games with in-app purchases Information Need Results.

Rank	System 1	System 2	System 3
1	R	R	N
2	N	R	N
3	R	R	N
4	N	N	N
5	R	R	N
6	R	N	N
7	R	R	R
8	R	R	R
9	R	N	R
10	R	R	R
P@10	0.80	0.7	0.4
AvP	0.747	0.852	0.282

C. Free games with in-app purchases

Information Need: Free to play games that contain in-app purchases.

Relevance Judgment: Here we intend to retrieve games that are technically free to play but have in-app purchases that might give you advantages in the game. We can find this information on the game categories but also on reviews from players that show their frustration from "pay-to-win" games, where the in-app purchases give you a clear advantage on the other players, making the game effectively not free.

Query: `("pay to win" OR "in-app purchase" ~10) AND free`

Conclusion: The first two systems had a much better performance than the third, as it can be seen in Table VII. In system 3 we assigned a bigger weight to the game description with the expectation that the publishers would specify that the game had in-app purchases, which turned out to not be the case, making the system less reliable. This query can make it hard to obtain relevant results as some reviews mention in various ways that the game is *not* pay-to-win, leading the system to return a non relevant document.

D. Games with a toxic community

Information Need: Games with a toxic community

Relevance Judgment: Here we intend to search for games where players consider the community toxic. In order to exclude other comments regarding the community we used the + operator to force a match with the *toxic* keyword. As publishers do not want the toxicity of the community associated with the game the only location where we can find this information is in the reviews text.

Query: `+toxic community`

Conclusion: As we can see from Table VIII the first two systems had a good performance. As expected the system 3 with a bigger weight to the review text gave us a perfect set of retrieved documents. So we decided to analyse how system 3 would behave if we reduced the review text weight to *0.5*. We expected a reduction of precision but we actually got 0 relevant documents retrieved, proving the importance of the review text for this Information Need.

TABLE VIII: Games with Toxic Community Information Need Results.

Rank	System 1	System 2	System 3
1	R	R	N
2	N	R	N
3	N	R	N
4	R	R	N
5	R	N	N
6	R	R	N
7	R	R	N
8	R	R	N
9	R	R	N
10	R	R	N
P@10	0.70	0.7	0.00
AvP	0.716	0.712	0.000

E. Fast Paced Games

Information Need: Games that offer a fast paced gameplay, sometimes described as "adrenaline-inducing games".

Relevance Judgment: Must not simply be a game with, per example, fast cars or fast races. The game must provide constant changing environments and be action-packed. Usually this information can be retrieved straight from the description, but reviews also portray this information quite well.

Query: `+fast pace`

Conclusion: All the 3 systems demonstrated good results, as it can be seen in Table IX, given the general correct use of the ‘‘fast pace” term in both game descriptions and games reviews. There is no real gain in falsely advertising games in this aspect since both fast and slow paced games are equally important in the game community, being a matter of personal taste.

TABLE IX: Fast Games Information Need Results.

Rank	System 1	System 2	System 3
1	R	R	R
2	R	R	R
3	R	R	R
4	R	R	R
5	R	R	R
6	R	R	R
7	R	R	R
8	R	R	R
9	R	R	R
10	R	N	R
P@10	1.00	0.9	1.00
AvP	1.000	1.000	1.000

F. Specific Game - Counter Strike

Information Need: Find information regarding a specific game, in this case Counter-Strike. This Information Need was introduced to study a navigational query type in our systems.

Relevance Judgment: Given the navigational nature, to be relevant, the first result must be the game document.

Query: `Counter Strike`

Conclusion: In the context of this Information Need, we changed the aforementioned weights and only considered the name field in the query. This had the expected result of retrieving the actual game document in the first result as expected, in each system.

TABLE X: Specific Game Information Need Results.

Rank	System 1	System 2	System 3
1	R	R	R
P@1	1.00	1.00	1.00
AvP	1.000	1.000	1.000

Overall Conclusions

Taking into account all the results from the multiple information needs experiments, we can calculate the MAP (*Mean Average Precision*) of the systems in order to compare them on a more general scope. Table XI contains the MAP values of each system. As we can see, System 2 improves on System 1 a little bit overall. However, System 3 is worse than the others. We knew this when deciding on the systems' properties, since we started out with a System 3 that weighed the reviews higher than the games' descriptions. However, this would make it really similar to System 1, since games have much more reviews than descriptions. Thus, we decided to test the opposite and confirm that the reviews indeed have a huge impact on the Information Needs, specially the ones of ***Informational*** type. We can also conclude that a system similar to System 2, but with weights configured to prefer reviews would probably be the best of both worlds, and provide the most accurate results.

TABLE XI: MAP values of the 3 system configurations.

System 1	System 2	System 3
0.814	0.859	0.357

XII. Information Retrieval Tool Evaluation

After this experiment, we have some remarks on the tool (Solr) as an Information Retrieval Tool. First, Solr lacks clear documentation and examples for many of its features and in all versions. Most of our trouble boiled down to lack of documentation on how to use Solr. Also, after fiddling with the dataset, we found out that working with nested documents is not trivial for retrieval operations (regarding retrieving both the parent and the child, when one or the other matches the query). In hindsight, since some of the members had some experience with ElasticSesarch, and even though some things are not trivial either with this tool, probably it should have been the selected tool, since experience and familiarity is a good starting point with these kinds of tools. Besides, the documentation and support is generally better, and there are more friendly and intuitive UIs that can be used with it, such as Kibana.

XIII. Semantic Web

Following its creators' desires, the World Wide Web (WWW) quickly became the main way of sharing information. In this new platform, users started to share many different types of information without a common structure, leading to an extremely heterogeneous environment. While humans could make sense of this chaotic environment and understand the relation between documents, the same did not apply to machines — the content was mostly non **machine-readable**, making it impossible to automatize some day-to-day mechanisms. The Semantic Web, as envisioned by Tim Berners-Lee [1], appeared to tackle this issue — make the Web machine-readable. As an extension of the current WWW, in Semantic Web, machines would understand the contents of one document and its integration/relation with others. This meant that machines had to interpret each document's semantics and draw inferences or even create new knowledge. To achieve this, the Semantic Web makes use of ontologies to characterize data. In the following sections, we semantically explore our data, identifying the existing entities and their relations (amongst themselves and with other existing ontologies).

XIV. Existing Ontologies

The Video Game Ontology (VGO) [17] was created by researchers at Lappeenranta University of Technology, Finland and Universidad Politecnica de Madrid, Spain, to capture knowledge about events that happen in video games and information about the player. It provides a way of describing concepts in the video game domain such as: the game itself, players, achievements, leaderboards, in-app purchases, the game genres, among others. While sharing some entities with our domain (*e.g*, the *Game* and *Genre* class), its main goal does not align with ours. We try to provide the user with information about video games that will help them in the decision-making process of purchasing a game. To do this, we provide the user with the game's reviews and properties such as the price, required age, positive ratings and number of owners. Nonetheless, the VGO will be useful when developing an ontology for our specific domain. The class *Game* is present in both ontologies and can be associated with the *owl:sameAs* property, thus connecting the same game in both ontologies resulting in the linking of two different data sources.

XV. Ontology Creation

To create the ontology for our domain we used the Protégé [16] tool, a open-source ontology editor that allows users to create, visualize and query ontologies.

Every ontology is composed of three main types of entities: classes, object properties and data properties. Classes represent concepts or types of things, and are used as an abstraction mechanism for grouping resources with similar characteristics. The first step in building our ontology was identifying which concepts could be considered ontology classes. We decided on 7 different classes, summarized in Figure 6, 5 represent the 3 main concepts of our domain: *Games*, *Reviews* and *Organizations* and 2 represent fixed characteristics of the game individuals: *Genres* and *Category*, defined as enumerated classes. The classes *Publisher* and *Developer* are sub-classes of the class *Organization* forming a non-disjoint generalization, where individuals can be of the *Developer* and *Publisher* class at the same time. Apart from the classes *Organization*, *Publisher* and *Developer* all the other classes were set as disjoint from all the ontology classes. In addition to this, the classes *Publisher* and *Developer* were specified as defined classes using the following existential restrictions, `published some Game` and `developed some Game` respectively.

Object properties in an ontology are entities that allow to link individuals to other individuals. The term individual, in an ontology domain, corresponds to an instance of a class. So, the next step to build the ontology was to define these properties. We started out by mapping some of the associations present in our Conceptual Model, Figure IV, that linked different classes and then defined the inverse for each one, summarized in Figure 6.

Next we set the characteristics of each object property, all the properties were set as asymmetric and irreflexive, as other characteristics such as symmetry, relexiveness and transitiveness do not apply to our domain. The properties **is_developed_by**, **is_published_by** and **refers_to** are functional, making their inverse properties, **developed**, **published**, and **is_reviewed_on** respectively inverse functional.

Data proprieties are the properties of an ontology that allow to link individuals of a class with data values. The data properties in our domain are manly the entities names' and descriptions' and numeric values associated with the *Games'* individuals such as, the number of owners, the price and the average playtime. All the data proprieties were set as functional.

XVI. Ontology Population

After the ontology was defined we started to research the tools available to populate it. Cellfie [2], a desktop plugin for importing spreadsheet data into OWL ontologies, was chosen because it is already integrated in Protégé and allows to process the spreadsheets and use cell referencing with an extension of the OWL 2 Manchester Syntax [6].

To import data with the Cellfie plugin it is required to have a different spreadsheet for each ontology class, as our original datasets were in the *csv* format, we only needed to extract the names of the *Genre* and *Category* individuals to a separate spreadsheet. Due to Protégé being a very resource heavy tool and not being able to handle huge amounts of data, like our original datasets, we decided to only import a subset of them. The ontology was populated with 20 *Organization* individuals, chosen at random, 150 *Game* individuals, each one with at most 10 related *Review* individuals, 3 *Category* individuals and 10 *Genre* individuals.

To parse the spreadsheet data and transform it into the previously defined ontology classes and their properties Cellfie allows to upload a set of rules, in a *json* file. Figure 7 and 8 show the set of rules used to import the *Game* and *Review* individuals, the two classes with the higher number of object and data properties.

Sheet Name	Start Column	End Column	Start Row	End Row	Rule
steam_store_small	A	S	2	+	Individual: @B* Types: Game Facts: hasAppID @B* Facts: hasAveragePlaytime @P* (xsd:integer) Facts: hasDescription @D* Facts: hasMedianPlaytime @Q*(xsd:float) Facts: hasName @C* Facts: hasNegativeRatings @O*(xsd:integer) Facts: hasNumberOfOwners @R* (xsd:integer) Facts: hasPlatform @J* Facts: hasPositiveRatings @N*(xsd:integer) Facts: hasPrice @S*(xsd:float) Facts: hasReleaseDate @F* Facts: hasRequiredAge @K*(xsd:integer) Facts: hasWebsite @E* Facts: isEnglish @G*(xsd:boolean) Facts: is_developed_by @H* Facts: is_published_by @I* Facts: is_of_genre @M* ({"[\s\S]+?\(?::)$"}) Facts: is_of_category @L* ({"(\s\S]+?\(?::)$"})

Fig. 7: Game rules.

Sheet Name	Start Column	End Column	Start Row	End Row	Rule
Sheet 1 - steam_revie	A	S	3	+	Individual: @A* Types: Review Facts: hasText @C* Facts: hasPositiveSentiment @O* (xsd:integer) Facts: hasNumberOfHelpfulVotes @E* (xsd:integer) Facts: refers_to @B*

Fig. 8: Review rules.

XVII. SPARQL Querying

With the ontology correctly defined and populated we focused on answering some of the retrieval tasks not possible during the Information Retrieval phase. In order to do this

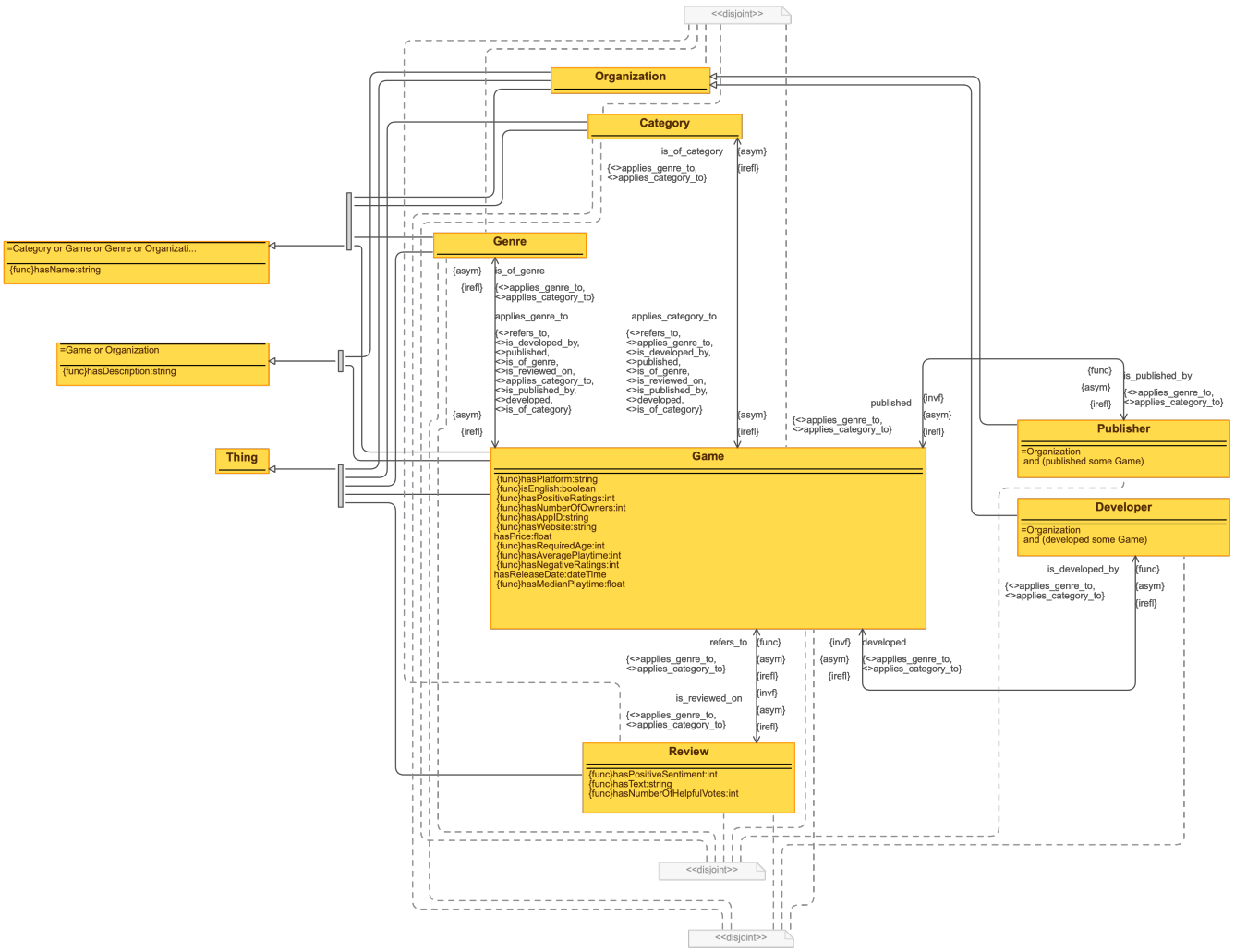

Fig. 6: Steam Game ontolgy.

we used the Semantic Web's main technology for querying data, SPARQL [9]. Some of the original queries suffered some alterations, due to the dataset subset used in the ontology population not returning meaningful results. The chosen retrieval tasks, respective SPARQL queries, results, and description of the modifications to the original query are presented next.

For the following SPARQL queries the set of prefixes in Listing XVII were used.

```
PREFIX owl: <http://www.w3.org/2002/07/
    owl#>
PREFIX rdf: <http://www.w3.org
    /1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/
    rdf-schema#>
PREFIX steam: <http://www.semanticweb.org
    /pedro/ontologies/2020/11/steam-games
    #>
```

Listing 2: SPARQL Query Prefixes

Question: How many games made by Valve have a minimum age under 18?

In this query we are trying to find how many games made by Valve have the minimum age less than 18 years old, which is a valid approach to find the non violent games. To do this we retrieve the games using the class **Game** and filter using the object property **developed** and data property **hasRequiredAge**, both from the class Game, based on being made by Valve and having the minimum age less than 18. The SPARQL query for this question can be consulted in Listing XVII and its results in Figure 9.

```
    SELECT (count(?games) AS ?num)
    WHERE {
        ?games a steam:Game .
        ?games steam:is_developed_by
            steam:Valve .
        ?games steam:hasRequiredAge ?age.
        FILTER(?age <18) .
    }
```

Listing 3: “How many games made by Valve have a minimum age under 18?” SPARQL Query.

num
"25"^^<http://www.w3.org/2001/XMLSchema#integer>

Fig. 9: Results for “How many games made by Valve have a minimum age under 18?”.

Question: Which are the top 10 most profitable games?

Here we try to find out which games are the most profitable of all time, in order to do this, we multiply the estimated number of owners by the game’s price, both values given by data properties of the class *Game*. Then, we order all the *Game* individuals by the results of the multiplication and limit the results to the first 10. The SPARQL query for this question can be consulted in Listing XVII and its results in Figure 10. The obtained results do not represent the most profitable games of the whole Steam Store, but rather of the subset of 150 games imported into Protégé.

```
SELECT ?name WHERE {
        ?x a steam:Game .
        ?x steam:hasName ?name .
        ?x steam:hasPrice ?price .
        ?x steam:hasNumberOfOwners ?
           owners .
        BIND ((?price*?owners) AS ?result
           )
}
ORDER BY DESC (?result)
LIMIT 10
```

Listing 4: “Which are the top 10 most profitable games?” SPARQL Query.

name
"Portal"
"Counter-Strike: Condition Zero"
"Counter-Strike"
"Half-Life 2"
"Portal 2"
"Counter-Strike: Source"
"Left 4 Dead 2"
"Garry's Mod"
"Half-Life 2: Deathmatch"
"Killing Floor"

Fig. 10: Results for “Which are the top 10 most profitable games?”.

Question: Which are the top 3 most played games of the last 8 years?

Here we try to analyse the most played games of the last 8 years, by ordering the *Game* individuals by their *hasAverage-Playtime* values. Then, like in the previous query, we make use of the **LIMIT** keyword to just return the top 3. Initially, we intended with this query to find out which were the top 3 most played games of the last 5 years, 2015 to 2020. However, the subset utilized to populate the ontology contains *Games* from 1998 to 2013, so we decided to change the number of years from 5 to 8 to obtain better results.

```
SELECT ?name ?playtime
WHERE {
        ?game a steam:Game .
        ?game steam:hasName ?name .
        ?game steam:hasAveragePlaytime ?
           playtime .
        ?game steam:hasReleaseDate ?date
           .
        FILTER (?date >= "2012-01-01"^^
           xsd:dateTime)
}
ORDER BY DESC (?playtime)
LIMIT 3
```

Listing 5: “Which are the top 3 most played games of the last 8 years?” SPARQL Query.

name	playtime
"Dota 2"	"23944"
"Counter-Strike: Global Offensive"	"22494"
"X Rebirth"	"1744"

Fig. 11: Results for “Which are the top 3 most played games of the last 8 years?”.

Question: Which are the top 3 organizations whose games have the best reviews?

Here we try to find out the top 3 **Organizations** that have the best reviews, *i.e.* reviews with the most helpful votes. We start by getting the list of all games and their respective organizations and number of helpful votes. Then, we group by **Organization** in order to obtain the sum of all the votes. Finally we sort the results, descending, and limit to the top 3.

```
SELECT  ?org (SUM(?votes) as ?totalVotes)
WHERE
{
        ?game a steam:Game.
        ?game steam:hasName ?name.
        ?game steam:is_published_by ?org.
        ?review a steam:Review.
        ?review steam:refers_to ?game.
        ?review steam:
           hasNumberOfHelpfulVotes ?votes
}
GROUP BY ?org
ORDER BY DESC(?totalVotes)
LIMIT 3
```

org	totalVotes
Valve	"204"^^<http://www.w3.org/2001/XMLSchema#integer>
IdSoftware	"64"^^<http://www.w3.org/2001/XMLSchema#integer>
PopcapGames	"57"^^<http://www.w3.org/2001/XMLSchema#integer>

Fig. 12: Results for "Which are the top 3 organizations whose games have the best reviews?".

top10AreFree
"false"

Fig. 13: Results for "Are the top 10 most played games free?".

Listing 6: "Which are the top 3 organizations whose games have the best reviews?" SPARQL Query.

Question: Are the top 10 most-played games free?

Here we try to find out if the top 10 most-played games are free. For this, we get the top 10 games based on average play-time and add the isFree flag, sorting them from the most played to the least played. We then count the number of free games in the list and if it matches the number of games (10), it means that the top-10 are all free, being false otherwise.

```
SELECT ?top10AreFree
WHERE {
        BIND( (?count =  10) AS ?
           top10AreFree)
        {SELECT (COUNT(?isFree) AS ?count
           )
        WHERE {
                FILTER (?isFree = true) .
                {SELECT ?isFree
                WHERE {
                ?game a steam:Game.
                ?game steam:hasName ?name
                    .
                ?game steam:
                   hasAveragePlaytime ?
                   playTime .
                ?game steam:hasPrice ?
                   price

                BIND( (?price =  0) AS ?
                   isFree)
                }
                ORDER BY DESC(?playTime)
                LIMIT 10}

        }
        GROUP BY ?isFree}
}
```

Listing 7: "Are the top 10 most played games free?" SPARQL Query.

Question: Which are the top 20 organizations (developers and publishers) in Action genre by average playtime?

Here we try to find out the top 20 **Organizations**, meaning they could be **Publishers** or **Developers**, that have the most playtime in their **Action** games, on average. We start by getting the list of **Action** games and their respective organizations and playtime. Then, we group by **Organization** in order to average the playtime of all their **Action** games. Finally we sort by playtime, descending, and limit to the top 20.

```
SELECT * WHERE {
        {SELECT  ?org (AVG(?playtime) AS
           ?avg_playtime)
        WHERE {

                ?game a steam:Game .

                { ?game steam:
                   is_published_by ?org
                   . }
                UNION
                { ?game steam:
                   is_developed_by ?org .
                    }

                ?game steam:
                   hasAveragePlaytime ?
                   playtime .
                ?game steam:is_of_genre
                   steam:Action .

        }
        GROUP BY ?org}
}
ORDER BY DESC(?avg_playtime)
LIMIT 20
```

Listing 8: "Which are the top 20 organizations (developers and publishers) in Action genre by average playtime?" SPARQL Query.

Question: What is the ratio of number of reviews to number of players?

With this query, we're trying to retrieve which games have more players than reviews, and this regularly means that it's a well-received game due to most reviews being about negative experiences with the game. With the nested select statement, we get the number of owners and reviews for each game, using the **Game** and **Review** class, we group by **Game** to perform the *count* of the reviews. In the outer select statement, the ratio calculation of reviews to the number of players is performed. Finally, we sort by ascending order to show the

org	avg_playtime
Valve	"3604.3"
Egosoft	"1358"
TripwireInteractive	"780"
Lucasarts	"664"
InfinityWard	"647.5"
ArkaneStudios	"634"
UnknownWorldsEntertainment	"627"
GscGameWorld	"493"
GearboxSoftware	"492.5"
DoubleFineProductions	"491"
Neversoft	"462"
TerminalReality	"441"
Activision	"421.4"
Ubisoft	"413.5"
TroikaGames	"350"
RavenSoftware	"326"
Sega	"312"
Monolith	"312"
MajescoEntertainment	"294"
MindwareStudios	"270"

Fig. 14: Results for "Which are the top 20 organizations (developers and publishers) in Action genre by average playtime?".

ones who should represent well-received games (more players than reviews). The SPARQL query for this question can be consulted in Listing XVII and its results in Figure 15. In this case, the number of reviews imported to Protégé was limited to 10, unlike the number of players, so it doesn't return interesting results due to the number of players being much higher.

```
SELECT ?games (((?num_revs*100)/?
    num_owners) AS ?ratio)
WHERE
{SELECT ?games ?num_owners (count(?revs)
    as ?num_revs)
    WHERE {
        ?games a steam:Game .
        ?games steam:hasNumberOfOwners ?
            num_owners .
        OPTIONAL{
            ?revs a steam:Review .
            ?revs steam:refers_to ?games
                .
        }
    }
GROUP BY ?games ?num_owners}
ORDER BY ASC(?ratio)
LIMIT 10
```

Listing 9: "What is the ratio of number of reviews to number of players?" SPARQL Query.

games	ratio
730	"0"^^<http://www.w3.org/2001/XMLSchema#decimal>
1670	"0"^^<http://www.w3.org/2001/XMLSchema#decimal>
4700	"0"^^<http://www.w3.org/2001/XMLSchema#decimal>
4780	"0"^^<http://www.w3.org/2001/XMLSchema#decimal>
3570	"0"^^<http://www.w3.org/2001/XMLSchema#decimal>
2320	"0"^^<http://www.w3.org/2001/XMLSchema#decimal>
4290	"0"^^<http://www.w3.org/2001/XMLSchema#decimal>
570	"0.000005"^^<http://www.w3.org/2001/XMLSchema#decimal>
440	"0.00002"^^<http://www.w3.org/2001/XMLSchema#decimal>
620	"0.00005"^^<http://www.w3.org/2001/XMLSchema#decimal>

Fig. 15: Results for "What is the ratio of number of reviews to number of players?".

XVIII. Evaluation

Regarding the ontology creation, the team found the process intuitive and straightforward. About the ontology population, Protégé limitations concerning the quantity of data proved to be a challenge as only a small subset of the project data could be used, making the process of retrieving results from queries hard to find interesting results. Lastly, about querying, several unexpected bugs also slowed the team's development, mainly some members not being able to use the main window of SPARQL having to fall back to an experimental window where essential functionalities were missing (e.g., nested statements, basic commands). The quality of the results was negatively impacted by the dataset size, as mentioned above, other than that, the team was able to circumvent the shortcomings of Protége and perform the initially considered queries.

XIX. Semantic Web vs. Information Retrieval

Information Retrieval (IR) and Semantic Web (SW) are two distinct areas that serve different, albeit related, purposes. The first allows the user to retrieve information with a simplified query - oftentimes a human-friendly string - which might return results from different types and contexts depending on its configuration. However, this does not mean that they will be connectable among them: that is provided in SW. With the latter, there is a series of predicates serving as meaningful links among instances. This allows the "machine" to understand the connections and make the search process faster if you know what you are looking for and how to query it in an SW-friendly way.

This presents us the main difference between both systems: If the user has an Information Need solely and is not certain what to look for or how to query about it, IR might be more appropriate; on the other hand, knowing the connections between the instances and the available predicates empowers the users in the sense that they can now place more complex questions, which is not usually the case.

In the end, it's all a matter of trade-offs. An IR system allows for simple questions like "What is the developer of Counter-Strike?" or "How many players have the Dota 2 game?" which are most common among users. SW, on the other hand, allows for more complex and often statistic-based questions like "Considering Valve's games, how many of their action games were played by at least 1 million users?" or "Is

there any free strategy game with more than 100 million users and a positive ratio of reviews?"

It's also relevant to note that both solutions can be integrated into existing knowledge collections by having specialized search engines or merging with existing ontologies, respectively for IR and SW.

To summarize, both Semantic Web and Information Retrieval have their use cases and should be leveraged appropriately to fulfill the multiple users' information needs.

XX. Conclusions and Future Work

In the paper's initial sections regarding data characterization, we presented the processed used to create our dataset. We enumerated the steps done to clean the data, as well the tools used, and showed how we were able to integrate different sources of information, achieving a better structured and more detailed dataset that can be easily integrated into a search system. We then presented our information retrieval process for the search system, evaluating the search tool to compare Solr and ElasticSearch through its different features and why Solr was selected as the best for our project. We defined a set of information retrieval tasks, our indexing process, the steps we took from the end of the data characterization step to the indexing process, the Solr schema and filters used to optimize the system. We selected six information needs for users from the previously defined retrieval tasks and performed a comparison between three systems, evaluating the performance through the precision from the first ten documents retrieved. Evidencing the impact of Solr filters and weighted search queries. We concluded that the correct usage of stemming and synonyms in the dataset (done in System 2) is very relevant to the accuracy of the Information System.

In the Semantic Web domain, we looked at our project from a different perspective, starting by exploring similar existing ontologies in our domain. After, we created, from scratch, an ontology to represent our system and the inner connections, which was queried, with SPARQL, to answer the remaining retrieval tasks. The results were as expected, within the limitations of the dataset size, and allowed us to compare the Information Retrieval and Semantic Web paradigms.

In future work, we will concentrate on reducing the high percentage of games whose organizations do not yet have a description in the system and designing, building and populating an ontology for the domain at hand.

References

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web: A New Form of Web Content That is Meaningful to Computers Will Unleash a Revolution of New Possibilities". In: *ScientificAmerican.com* (May 2001).

[2] *Cellfie*. URL: https://github.com/protegeproject/cellfie-plugin (Accessed Jan. 4, 2021).

[3] Nik Davis. *Steam Store Games Dataset*. 2019. URL: https://www.kaggle.com/nikdavis/steam-store-games (Accessed Nov. 24, 2020).

[4] *Elasticsearch replaced Solr as the most popular search engine*. URL: https://db-engines.com/en/blog_post/55 (Accessed Jan. 4, 2021).

[5] *Kaggle*. URL: https://www.kaggle.com (Accessed Nov. 24, 2020).

[6] *OWL 2 Web Ontology Language Manchester Syntax (Second Edition)*. URL: https://www.w3.org/TR/owl2-manchester-syntax/ (Accessed Jan. 4, 2021).

[7] Antoni Sobkowicz. *Steam Review Dataset*. 2017. URL: https://zenodo.org/record/1000885 (Accessed Nov. 24, 2020).

[8] *spaCy*. URL: https://spacy.io/ (Accessed Nov. 24, 2020).

[9] *SPARQL Query Language for RDF*. URL: https://www.w3.org/TR/rdf-sparql-query/ (Accessed Jan. 4, 2021).

[10] *Steam Reviews*. URL: https://store.steampowered.com/reviews/ (Accessed Nov. 24, 2020).

[11] *Steam Store*. URL: https://store.steampowered.com (Accessed Nov. 24, 2020).

[12] *SteamSpy*. URL: https://steamspy.com (Accessed Nov. 24, 2020).

[13] *The Apache Solr Website*. URL: https://lucene.apache.org/solr/ (Accessed Dec. 2, 2020).

[14] *The Elasticsearch Website*. URL: https://www.elastic.co/elasticsearch/ (Accessed Dec. 2, 2020).

[15] *The OpenRefine Website*. URL: https://openrefine.org (Accessed Nov. 24, 2020).

[16] *The Protege Website*. URL: https://protege.stanford.edu/ (Accessed Jan. 11, 2021).

[17] *The Video Game Ontology*. URL: http://vocab.linkeddata.es/vgo/ (Accessed Dec. 28, 2020).

[18] *wikibase-sdk NPM package*. URL: https://www.npmjs.com/package/wikibase-sdk (Accessed Nov. 24, 2020).

[19] *Wikidata*. URL: https://www.wikidata.org/wiki/Wikidata:Main_Page (Accessed Nov. 24, 2020).

[20] *Wikipedia*. URL: https://en.wikipedia.org/w/api.php (Accessed Nov. 24, 2020).

[21] *wtf_wikipedia NPM package*. URL: https://www.npmjs.com/package/wtf_wikipedia (Accessed Nov. 24, 2020).

[22] *Zenodo*. URL: https://www.zenodo.org (Accessed Nov. 24, 2020).