Steam Game Data Search Engine: Search System

Bruno Mendes
up201906166@edu.fe.up.pt
M.EIC
Faculty of Engineering of the University of Porto
Portugal

Fernando Rego
up201905951@edu.fe.up.pt
M.EIC
Faculty of Engineering of the University of Porto
Portugal

Joel Fernandes
up201904977@edu.fe.up.pt
M.EIC
Faculty of Engineering of the University of Porto
Portugal

ABSTRACT

Steam is a video game digital distribution service and storefront by Valve. It is composed of a large game library from diverse genres and categories, which are often hard to find. This project aims to prepare and process data collected from publicly available *Steam API's* and *steamspy.com*, to fulfill the information needs of a user looking for a new game. The prepared dataset, after cleanup and feature engineering tasks, is the source of knowledge to fulfill a user's information needs through querying, which yields results that prove that specialized and more verbose queries lead to more relevant results.

CCS CONCEPTS

• **Information systems** → **Information retrieval**; **Structured text search**.

KEYWORDS

datasets, information processing, information retrieval, full-text search, steam, search engine

1 INTRODUCTION

According to Steam's website [2], Steam is the ultimate destination for playing, discussing, and creating games. In other terms, it is a large library for discovering and managing games. It is expected that users find new games with ease, even if they do not know their name, which is not always the case with the company's own search engine.
To solve that issue, we developed a search engine starting with data retrieved from the publicly available *Steam's API's*, *steamspy.com* and the *Wikipedia* API. The retrieved dataset is analyzed and characterized in 2.1. It it then processed to the point it is refined, properly indexed and ready to be queried, which we describe in 2.2. The system is then queried according to the identified information needs in 3.3, producing results that are evaluated in 3.4.
The implementation of proper user-friendly search system, able to generate queries based on user's hints, is discussed in 4.

2 M1 - DATA PREPARATION

The first milestone is focused on the collection and processing of data, aimed at feeding the search engine.

2.1 Data Source - Identification and Characterization

The dataset used in this project was obtained from the Data World [15], where an enormous variety of open data is collected, and includes data from a total of 13,357 games in a CSV format file with a size of 52.6 MB. The dataset was generated and made available open-source, with MIT License, by CraigKelly [10] combining public data from Steam API [14] and Steam Spy [13].
Due to the lack of text fields, additional description data from each game queried by title was taken from the Wikipedia API [6] using a web client developed by the team. The textual data was appended (in the form of a new column named *WikiData*) to the original CSV.
After an initial phase of data analysis, the team came across 149 duplicated games, fields with high rate of missing values as we show in Table 1, some fields that can be simplified into one and irrelevant or inconsistent attributes as we show in Figure 1.

Field	Missing	%
`ReleaseDate`	87	0.65
`PriceCurrency`	2,618	19.6
`SupportEmail`	3,518	26.34
`SupportEmail`	5,204	38.96
`AboutText`	662	4.96
`Background`	701	5.25
`ShortDescrip`	1,862	13.94
`DetailedDescrip`	658	4.93
`DRMNotice`	13,272	99.36
`ExtUserAcctNotice`	13,202	98.84
`LegalNotice`	7,812	58.49
`Reviews`	10,043	75.19
`SupportedLanguages`	35	0.26
`Website`	3,268	24.47
`PCMinReqsText`	728	5.45
`PCRecReqsText`	7,555	56.56
`LinuxMinReqsText`	10,293	77.06
`LinuxRecReqsText`	12,151	90.97
`MacMinReqsText`	8,725	65.32
`MacRecReqsText`	11,619	86.99

Table 1: Missing values per field in the original dataset.

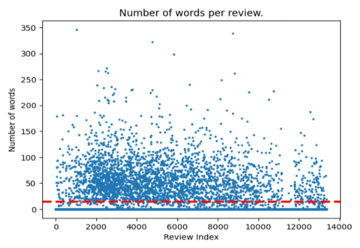


Figure 1: Number of words per review. Average of 14.2 words.

2.2 Dataset preparation

Due to the aforementioned problems, we need to process the dataset in order to obtain a more suitable and cleaner dataset for modeling.

2.2.1 Data cleaning. The first step of the applied pipeline involved the removal of duplicates by *ResponseID*, which are unique to each game.
Later, we deleted the columns *QueryID* and *QueryName* since they are repeated as *ResponseID* and *ResponseName*. We also deleted repeated rows with the same *ResponseID*.
We deleted some counters that we found irrelevant to our future engine, such as *AchievementHighlightedCount*, *ScreenshotCount*, *MovieCount* and *PackageCount*. These are very specific information about a game that a user will most likely not know or not care about when looking for a new game.
We deleted some flags - *PCReqsHaveMin*, *PCReqsHaveRec*, *LinuxReqsHaveMin*, *LinuxReqsHaveRec*, *MacReqsHaveMin*, *MacReqsHaveRec* -, which are easily derived from the length of the requirements for each of these platforms.
Since there were some columns that included links, we deleted those as well - *SupportEmail*, *SupportURL* and *Background*. There were some columns with legal information related to the game itself: *LegalNotice*, *DRMNotice* and *ExtUserAcctNotice*. We got those removed, too.
We deleted the column that contained the *PriceCurrency* the game was sold at since this value is the same in all rows.
Redundant data related to the games description and details - *AboutText*, *ShortDescrip* - are also removed since these are overshadowed by the more complete *DetailedDescrip*, which got renamed to *PromotionalDescription* to accomodate the *WikiData* longer description or story.
We removed variance data related to the number of players and owners the game has (*SteamSpyOwnersVariance* and *SteamSpyPlayersVariance*), which would be tailored for a predictive model in a data mining project but not quite for text-based retrievals.
We removed the *Reviews* column due the complexity required to parse it and extract relevant information from it: most rows had it empty and the ones that did not were messy, with grammar mistakes and low quality input.

2.2.2 Feature engineering. We started by grouping categories, features and genres. These values are displayed as boolean mask columns in the original dataset, which is not great when it comes to memory usage. Due to this we grouped them and created a new column for each of these topics that includes a list of the value that are True.
Each game has (potentially) different text and voice languages. We converted the *SupportedLanguages* column into two, representing the two means of communication with the end user.

2.2.3 Conversion to JSON. JSON (JavaScript Object Notation), although less compact than CSV, brings a stricter notion of types and a clearer syntax for list-like attributes, such as our languages and genres.

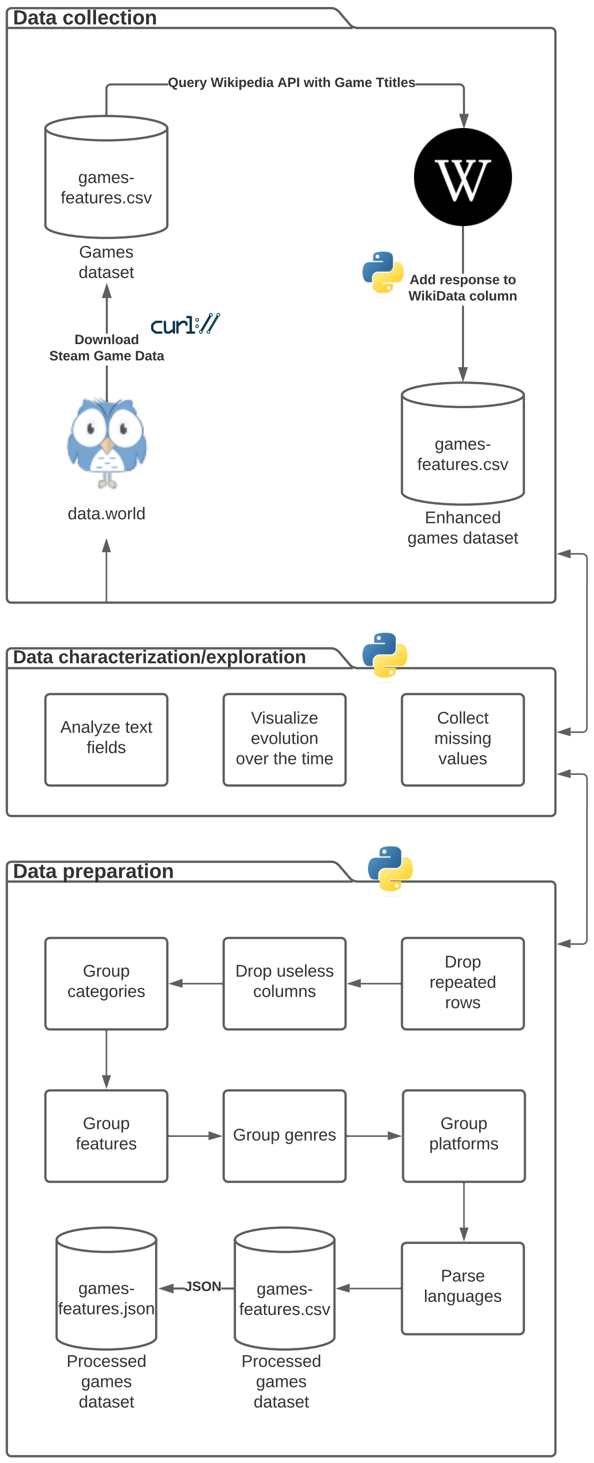


Figure 2: Pipeline datagram.

2.3 Dataset characterization

After completing all steps of data preparation, we obtained a dataset `processed.json` with 13,208 entries and a size of 44.9 MB. It has the fields described in Table 2. The conceptual model for the dataset is presented in Figure 3.

Table 2: Fields of the final dataset.

Field	Type	Description
`Name`	Text	Game name
`Release Date`	Date	Game release date
`Promotional Description`	Text	Description of the game
`Wiki Data`	Text	Wiki data retrieved from the Wikipedia API [6]
`Categories`	List	List of categories related to the game
`Genres`	List	List of genres related to the game
`Is free`	Boolean	True if the game is free, False otherwise
`Initial and Final Price`	Real	Initial and final price of the game in USD
`Free Version Available`	Boolean	True if the there is a free version of the game
`Platforms`	Text	Platforms available for the game (Windows, Linux and Mac)
`PC Requirements`	Text	PC Requirements for each platform (Windows, Linux and Mac)
`Required Age`	Integer	Minimum required age
`Game Players and Owners`	Integer	Number of game owners and number of people who have played the game retrieved by Steam Spy [13]
`Metacritic`	Integer	Game metacritic score
`Recommendation Count`	Integer	Number of game recommendations
`Counters`	Integer	Counters of game DLCs, game Demos, developers and publishers
`Achievements Count`	Integer	Number of game achievements
`Controller Support`	Boolean	True if the game supports controller, False otherwise
`Website`	Text	URL to the official website of the game
`Header Image`	Text	URL to the header image of the game
`Supported Languages`	Text	Supported game languages and audio languages

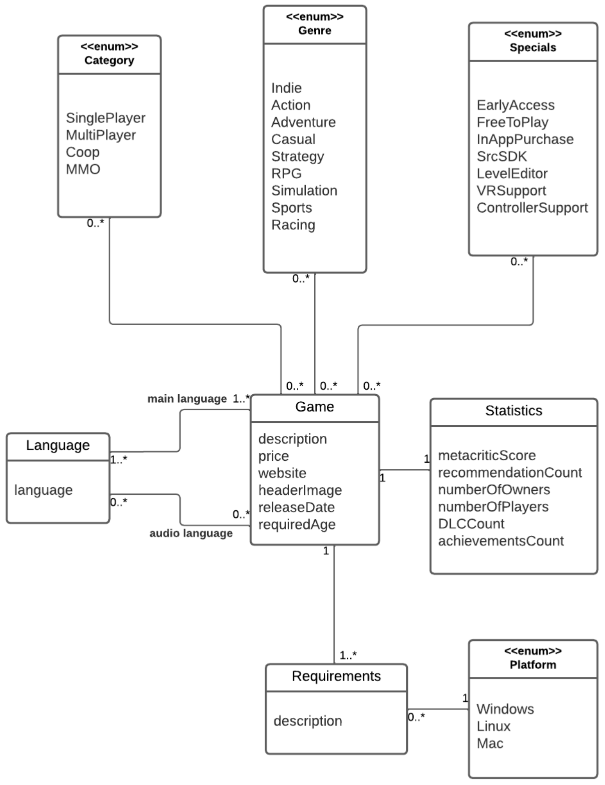


Figure 3: Conceptual model.

2.4 Data exploration

The processed dataset is more consistent and dense when it comes to the presence of data. About the latter, we should have in mind:

- Most games specify the minimum requirements without higher, recommended ones. This is to be expected, since most modern games already set a fairly high minimum spec requirements for smooth operation.
- Linux and Mac requirements are missing much more than the Windows counterparts: this is also expected, since most games are Windows-only.
- We do not find the Wikipedia API result eligible if it is a disambiguation page (we can't really tell what is the matching entry) or if the page is not categorized as a game page (to remove the possibility of a name being titled after a person or any entity with a Wikipedia page of their own). There is also the obvious possibility that Wikipedia does not have information about the queried game: in that case, the search engine should fallback to Steam's promotional description.

The Table 3 shows the missing values of the final dataset which confirms the previous conclusions that we should have in mind about the processed dataset.

Table 3: Missing values per field in the processed dataset.

Field	Missing	%
ReleaseDate	87	0.66
PromotionalDescription	658	4.98
Website	3,232	24.47
PCMinReqsText	728	5.51
PCRecReqsText	7,444	56.36
LinuxMinReqsText	10,157	76.9
LinuxRecReqsText	12,008	90.91
MacMinReqsText	8,611	65.2
MacRecReqsText	11,481	86.92
WikiData	10,946	82.87

There are also some interesting observations that will be of great importance later on for deciding on their importance for the documents rankings. We describe them in the following subsubsections.

2.4.1 Text analysis. From the Figure 4 we can easily notice that words such as *game*, *play* and *player* are, as expected, part of the most used vocabulary for our text fields. Without knowing, at this stage, which algorithms will come into play for keyword matching, we can already prospect that our system should not rank documents too much high based on the appearance of frequent words.

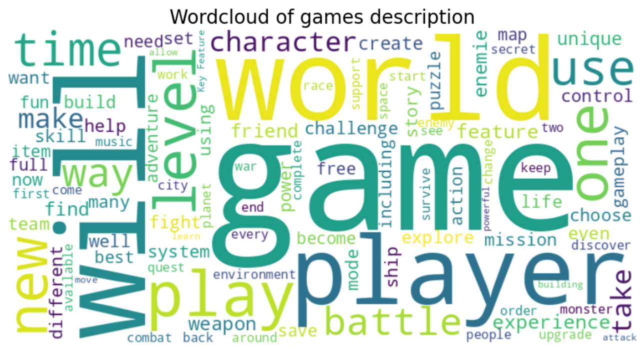


Figure 4: Word cloud for Steam's promotional descriptions.

2.4.2 Release date evolution. After analysing Figure 5 we have noticed that most games in the dataset are released near the date in which the Steam API was queried. That is probably related to cleanups of unpopular old games in Steam's database; it is definitely something to keep in mind when retrieving old games in the future.

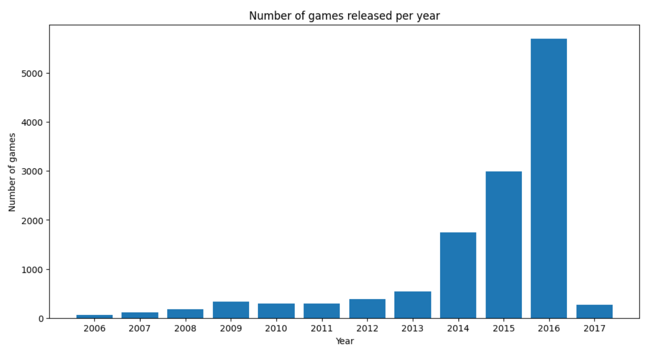


Figure 5: Number of games released per year.

2.4.3 Metacritic score evolution. Most of the games are received in a average way by the critics, which should make finding best games a feasible task. The Figure 6 shows that scores median seems to be lowering slightly over the years, which can be explained by the already explained theory of old unpopular game removal or by a lack of creativity by game creators.

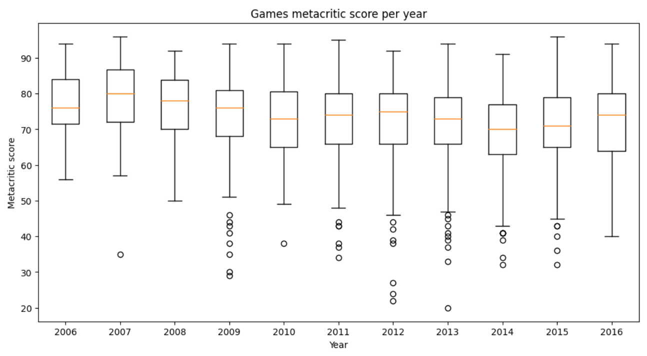


Figure 6: Game critics scores by year.

2.5 Information needs

When looking for a game, a user might be interested in fulfilling certain requisites, such as hardware support or cost; most importantly, though, is the thematic nature of the game. With our variety of text fields and precise filters, one can expect to get information such as:

- Which are the most recent action games with a large player base?
- What are the best games based on Vietnam?
- Is there any sport game that does not involve soccer?
- Are there cheap (<5$) games involving adventures in the desert?
- Are there games with more than 10 achievements involving zombies?
- What are the best games to solve riddles and mysteries?

3 M2 - INFORMATION RETRIEVAL

The second milestone focuses on the collection and indexing of data on the search engine, the information retrieval process and its evaluation. Due to the project's maturity and open-source nature, Apache Solr [3] was used as a search platform.

3.1 Data importing

From the previous stage, data was stored in JSON format. A game translates directly to a document:

```
{
    "ResponseID": 10,
    "ResponseName": "Counter-Strike",
    "ReleaseDate": "2000-11-01",
    "RequiredAge": 0,
    "DemoCount": 0,
    "DeveloperCount": 1,
    "DLCCount": 0,
    "Metacritic": 88,
    "RecommendationCount": 68991,
    "PublisherCount": 1,
    "SteamSpyOwners": 13033334,
    "SteamSpyPlayersEstimate": 9140731,
    "AchievementCount": 0,
    "ControllerSupport": false,
    "IsFree": false,
    "FreeVerAvail": false,
    "PurchaseAvail": true,
    "SubscriptionAvail": false,
    "PlatformWindows": true,
    "PlatformLinux": true,
    "PlatformMac": true,
    "PriceInitial": 9.99,
    "PriceFinal": 9.99,
    "PromotionalDescription": "Play the worlds number
        1 online action game. (...)",
    "HeaderImage": "http://cdn.akamai.steamstatic.com/
        steam/apps/10/header.jpg?t=1447887426",
    "SupportedLanguages": [
            "English",
            "French",
            "German",
            "Italian",
            "Spanish",
            "Simplified Chinese",
            "Traditional Chinese",
            "Korean"
        ],
    "Website": "",
    "PCMinReqsText": "Minimum: 500mhz processor (...)",
    "PCRecReqsText": "",
    "LinuxMinReqsText": "Minimum: Ubuntu 12.04 (...)",
    "LinuxRecReqsText": "",
    "MacMinReqsText": "Minimum: OS X 10.6.3 (...)",
    "MacRecReqsText": "",
    "wikiData": "Counter-Strike (CS) is a series of
        multiplayer tactical first-person shooter
        video games (...)",
    "Categories": ["Multiplayer"],
    "Features": [],
    "Genres": ["Action"],
    "SupportedAudioLanguages": []
    },
```

Figure 7: An example document shortened.

The simplest way to start working with this data in Solr was to create a collection and then feed it. We resorted to the simple Solr CLI to achieve this.

```
    solr create_core -c steam-games
    bin/post -c steam-games /data/steam-games.json
```

3.2 Indexing schema

Although Solr is able to automatically apply a set of operations provided it correctly identifies field types (which is easier with JSON), such as tokenizing a text field on querying and indexing time so that sole words match, we found that it was useful to build a customized schema to fine tune the search behavior.
For uploading a custom schema, we used the Solr REST API. It is necessary to perform this step before populating the collection, for it to be indexed properly.

```
curl -X POST -H 'Content-type:application/json' \
--data-binary @/data/schema.json \
http://localhost:8983/solr/docs/schema
```

Figure 8: Adding upon the default schema via the REST API.

An example of a field type in our schema is as follows:

```
{
    "name": "short_text",
    "class": "solr.TextField",
    "indexAnalyzer": {
        "charFilter": {
            "class": "solr.MappingCharFilterFactory",
            "mapping": "mapping-FoldToASCII.txt"
        },
        "tokenizer": {
            "class": "solr.ClassicTokenizerFactory"
        },
        "filters": [
            {"class": "solr.ClassicFilterFactory"},
            {"class": "solr.KStemFilterFactory"},
            {
                "class": "solr.PhoneticFilterFactory",
                "encoder": "DoubleMetaphone"
            },
            {"class": "solr.LowerCaseFilterFactory"},
        ]
    },
    "queryAnalyzer": (...)
}
```

Figure 9: The short_text field type shortened.

This field type converts, at index time, various characters to their equivalents in the UTF-8 charset. It proceeds by splitting the words into tokens, lower casing the characters, removing some english language artefacts (such as the possessive case), stemming the words (which includes resorting to their lexical base) and adding phonetic resemblance tokens. At query time, similar operations are performed, so that the resulting tokens match the indexed ones.

Table 4: Short text field as analysed by our system.

Input text	Index result	Query result
`Counter-Strike`	kntr counter strk strike	counter strike
`Tough Armors`	tf tough armr armor	tough armor
`Gangster's Paradise`	knks gangster prts paradise	gangster paradise

For longer text fields, such as WikiData, we use a stronger stemming strategy and resort to a custom synonym list, taken from an open source [1].

Table 5: Large text field as analysed by our system.

Input text	Index result	Query result
`I love football!`	i on 1 an love belov dear dearest honei footbal soccer game	i love football
`amusing bowling game`	amus bowl game	amus bowl game
`answer my phone arcade`	answer repli respond respons my phone telephon ring arcad up	answer my phone arcad

The default Solr schema for new cores tends to allow multi-value fields, outputting single-valued lists for integer ids or booleans. This does not affect search results; however, to improve memory and time efficiency, we specified the correct types in the schema. The complete schema additions can be seen in annex A.

3.3 Queries

Solr's query system purpose is to fulfill the information needs presented in 2.5, which are directly translated into eDisMax queries.

(1) Which are the most recent action games with a large player base?

```
Genres:action
q.op= AND
fq= [
    ReleaseDate:[2015-12-31T23:59:59Z TO *],
    SteamSpyPlayersEstimate:[100000 TO *]
    ]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))
```

(2) What are the best games based on Vietnam?

```
ResponseName:"Vietnam"~2^4,
PromotionalDescription:"Vietnam"~2,
WikiData:"Vietnam"~2^3
q.op= OR
fq= [Metacritic: [50 TO 100]]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))
```

(3) Is there any sport game that does not involve soccer?

```
ResponseName:"sports"~1^4
PromotionalDescription:"sports"~1
WikiData:"sports"~1^3
Genres:"Sports"
q.op= OR
fq= [
    ResponseName:(!soccer),
    PromotionalDescription:(!soccer),
    WikiData:(!soccer)
    ]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))
```

(4) Are there cheap (<5$) games involving adventures in the desert?

```
ResponseName:"desert"~2^4
PromotionalDescription:"desert"~2
WikiData:"desert"~2^3
q.op= OR
fq= [
    Genres:Adventure,
    PriceFinal:[* TO 4.99]
    ]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))
```

(5) Are there games with more than 10 achievements involving zombies?

```
ResponseName:"zombies"~2^4
PromotionalDescription:"zombies"~2
"WikiData:"zombies"~2^3
q.op= OR
AchievementCount:[10 TO *]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))
```

(6) What are the best games to solve riddles?

```
ResponseName:"riddles"~2^4
PromotionalDescription:"riddles"~2
wikiData:"riddles"~2^3
q.op= OR
fq = [Metacritic: [50 TO 100]]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))
```

For each query, we decided to give the biggest boost to *ResponseName* as it is the title of the game, followed by *Wikidata*, which has a smaller boost. The *PromotionalDescription*, unlike *Wikidata*, does not have any additional boost because after comparing both fields we come to the conclusion that the *Wikidata* has a better description about the game.
The boost function (*bf*) is thoroughly used to rank the documents according to their critics score and player base. This function uses the metacritic score where the higher the score the better the game is classified, the estimated number of players where games with high player numbers are expected to be searched more often and the number of recommendations which is usually an important factor for the choice of a game. Since the metacritic score ranges from 0 to 100 while the number of players can reach a very high value (>100000), the boost function uses logarithmic functions so that these values are balanced to obtain better results.

3.4 Result evaluation

Queries in 3.3 take advantage of some of Solr's query parser features such as fuzziness, proximity searches and field boosts, increasing the number of returned results. They also make sure to boost document or query fields using eDisMax features such as the boost function (*bf*) parameter.
Evaluating query results is not an easy task. For starters, it's hard to justify what are the most relevant results for a user: when they look for 'football', are they expecting to receive results related to American football or the rest-of-the-world football? On the other hand, it's tough, if not impossible, to manually judge all 10 000 + documents' relevance for a certain query.
To tackle these issues, we'll consider that the all relevant documents will be returned in the first 50 results for that query, from which we build a list of expected results and match it to the first 10 results yielded from that query. This number was fixed based on the fact that people tend to not even go through Google's second page [12]), so the system needs to yield very pertinent first results.
All measures taken in this section are automated through Python scripts. To showcase the importance of field boosts, we will compare the queries in 3.3 with our custom schema against their simpler, Lucene counterparts, with the default Solr schema.
We evaluated all the queries with their simpler counterparts using these measures:

- Average Precision (AvP): the average of the average of the precision values obtained for the set of top N documents existing each time a new relevant document is retrieved.
- Precision at *n* (P@n): fraction of the *n* first search results that are relevant.
- Recall at *n* (R@n): the fraction of retrieved relevant documents, i.e., what proportion of all the relevant documents has been retrieved in the first *n* search results.
- F1-Score at *n* (R@n): the harmonic mean of precision and recall at *n*.

These concepts and methodology are further explained in most search engine evaluation authorities [11].

Table 6: Query 1 - Information retrieval effectiveness measures.

Metric	Query Simple	Query Tuned
Average Precision (AvP)	0.37	0.79
Precision at 10 (P@10)	0.40	0.60
Recall at 10 (R@10)	0.67	1.00
F1-Score at 10 (F@10)	0.5	0.75

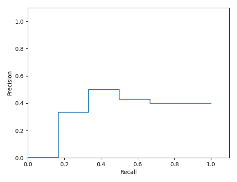


Figure 10: Query 1 - Precision-Recall Curve for the Lucene Query.

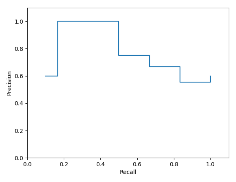


Figure 11: Query 1 - Precision-Recall Curve for the eDisMax Query.

Table 7: Query 2 - Information retrieval effectiveness measures.

Metric	Query Simple	Query Tuned
Average Precision (AvP)	0.50	0.64
Precision at 10 (P@10)	0.3	0.6
Recall at 10 (R@10)	0.5	1.00
F1-Score at 10 (F@10)	0.50	0.75

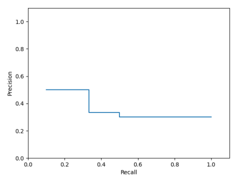


Figure 12: Query 2 - Precision-Recall Curve for the Lucene Query.

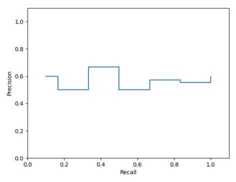


Figure 13: Query 2 - Precision-Recall Curve for the eDisMax Query.

Table 8: Query 3 - Information retrieval effectiveness measures.

Metric	Query Simple	Query Tuned
Average Precision (AvP)	0.25	0.54
Precision at 10 (P@10)	0.20	0.50
Recall at 10 (R@10)	0.29	0.71
F1-Score at 10 (F@10)	0.24	0.59

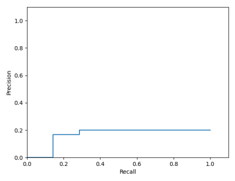


Figure 14: Query 3 - Precision-Recall Curve for the Lucene Query.

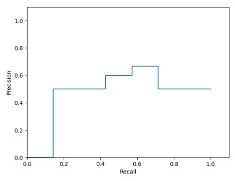


Figure 15: Query 3 - Precision-Recall Curve for the eDisMax Query.

Table 9: Query 4 - Information retrieval effectiveness measures.

Metric	Query Simple	Query Tuned
Average Precision (AvP)	0.31	0.57
Precision at 10 (P@10)	0.20	0.30
Recall at 10 (R@10)	0.33	0.50
F1-Score at 10 (F@10)	0.25	0.38

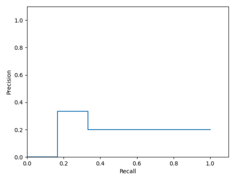


Figure 16: Query 4 - Precision-Recall Curve for the Lucene Query.

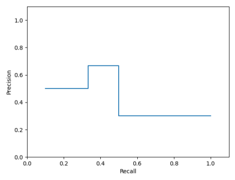


Figure 17: Query 4 - Precision-Recall Curve for the eDisMax Query.

Table 10: Query 5 - Information retrieval effectiveness measures.

Metric	Query Simple	Query Tuned
Average Precision (AvP)	0.07	0.64
Precision at 10 (P@10)	0.20	0.40
Recall at 10 (R@10)	0.33	0.67
F1-Score at 10 (F@10)	0.25	0.50

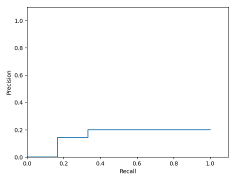


Figure 18: Query 5 - Precision-Recall Curve for the Lucene Query.

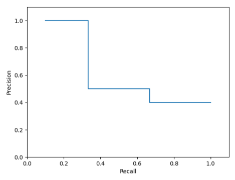


Figure 19: Query 5 - Precision-Recall Curve for the eDisMax Query.

Table 11: Query 6 - Information retrieval effectiveness measures.

Metric	Query Simple	Query Tuned
Average Precision (AvP)	0.35	0.64
Precision at 10 (P@10)	0.20	0.50
Recall at 10 (R@10)	0.29	0.71
F1-Score at 10 (F@10)	0.24	0.59

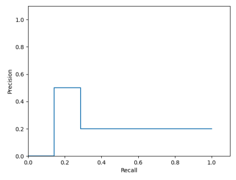


Figure 20: Query 6 - Precision-Recall Curve for the Lucene Query.

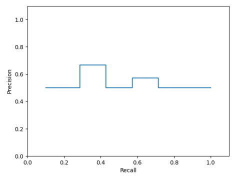


Figure 21: Query 6 - Precision-Recall Curve for the eDisMax Query.

As expected, defining a custom schema, namely for processing text fields in a more comprehensive manner, and using independent relevance boosts resulted in a considerable improvement.
It is also worth noting that some of the evaluated queries are, by nature, harder to respond to. For example, query 4 is meant to be interpreted as a search for games based in the actual, physical desert; however, expressions such as “desert house” match the keyword too. It is not clear how the search system should go around this.
The mean average precision for all the tested queries is coherent with the individual results.

```
Mean Average Precision (simple): 0.3083
Mean Average Precision (tuned): 0.6367
```

4 M3 - SEARCH SYSTEM

The third milestone is focused on the improvement of the previous search system, in an effort to make it more user-friendly. To make this possible, we worked on gathering more textual data based on textual entities and developing a facade for the search system that is able to generate Solr queries based on the user’s natural language hints, i.e. how much experienced we think the user might be as a gamer. As a last touch, related games to each query results are also retrieved, based on textual fields token correspondences.
This functionality is showcased in a custom command line interface that interfaces with a Solr local (or remote, if needed) daemon to automate the described mechanisms. Its interface can be seen in more detail in B.

4.1 Entity Exploration

Using spaCy[8] and their largest English pipeline package, we processed every *WikiData* field and extracted relevant entities: locations, names of people and names of organizations. The main objective was to get information related to where the games happened, names of the characters and the organizations involved in its making.
After extracting such information, we used the same script that generated the *WikiData* field (although with some performance improvements, e.g. concurrency) to search these terms using the Wikipedia API[6] and integrate everything into a new field *EntityWikiData*, that is simply the concatenation of each of the found entities' Wikipedia description.
Finally, we also made support for this new field on our schema with the *Large text* field type.
For example, for the game *Counter-Strike* released in 2000, we found the following entities: *Counter-Strike: Source, Jess "Cliffe" Cliffe, Minh, Ritual Entertainment, Counter-Strike Nexon: Studio, Counter-Strike Neo, Valve, Counter-Strike: Condition Zero, Hidden Path Entertainment, Xbox 360, the Counter-Strike Online, Half-Life, Turtle Rock Studios.* It is important to note that most of these items do not have a Wikipedia[5] page so they will be skipped when generating the new column.
The new column is a large one with raw information and is not supposed to be displayed to the user. It is the less boosted field on the *qf* and it is hidden on the CLI results.

4.2 New Signals

Google made personalized search a reality for everyone in 2009 [9]. Since then, most of us expect that a search engine knows about the user enough that it gives relevant results to that person. In our context, it is very difficult to do that. We can't make assumptions about the user's habits and we do not even know their last searches or clicks. This functionality could have been implemented by deploying a public product for some time and saving personal data in a database, which is not ideal in the academic environment. Instead, we took an approach that tries to identify which type of user the system is dealing with simply by analysing their query in natural language. In this context of games, we are trying to know how experienced the user is. The query analysis is as follows:

(1) Count the number n of words
(2) Count the occurrences k of "gamer terms", i.e. the number of words that match ones in a "gamer dictionary" [7]
(3) As k increases, boost the *RecommendationCount* and *Metacritic* and reduce the *SteamSpyPlayersEstimate* importance in the boost function
(4) As n increases, boost the *Metacritic* importance in the boost function
(5) If k is greater or equal than 2, switch the query operator to *AND* (from the default *OR*)

The rationale for this methodology is that longer (and more specific) queries are likely to be made by more experienced users with domain knowledge, i.e. longer-time gamers, which might be looking for higher-quality games instead of popular ones, more demanded by novices. That is why the *Metacritic* score receives a generous boost and the *RecommendationCount* a reduction.
The last step switches the query operator to *AND* when 2 or more "gamer terms" are written in a query. We can call this the "gamer mode", a stricter version of the system that discards all results that do not contain all terms of the query. This is very likely to discard very popular games which are not an exact match, which is harmful to the novice user, but useful to the experienced one looking to fulfill a very specific information need.
Let us see this analyser in action for three different queries, conceptually related, in table 12.

Table 12: Different signals for different queries.

Query	Metacritic *bf* weight	q.op	First result
shots	0.55	OR	Counter-Strike
shooter	0.65	OR	Counter-Strike
open world shooter	0.85	AND	Call of Duty

The first query, *shots*, is likely to be made by a novice. *shots* is not in the "gamer" dictionary and thus the *Metacritic* is not much boosted. The returned result is a popular game that the user is expecting.
The second query, *shooter*, includes a more precise, although generic, term that is indeed in the "gamer" dictionary. The returned games shall be more recommended by the critics and less popular, but the first result is the same.
The last query, *open world shooter*, represents a very specific information need and includes two terms in the dictionary: *open world* and *shooter*. This triggers the "gamer mode", i.e. the *AND* query operator, forcing *open world* to appear in the documents' textual fields. The returned result is by consequence a very precise one: contrary to *Counter-Strike: Global Offensive*, which is based on small, multiple maps, *Call of Duty: World at War* has no boundaries for the player to say locked in while in battle.

4.3 More Like This

We made use of the More Like This [4] Solr's[3] feature which queries similar documents in the collection, comparing chosen fields.
In our case, we used the default settings and our textual fields to do the similarity check with the same boost function we had for our queries.
For example, for the game *Counter-Strike* released in 2000, we found the following related game results: *Counter-Strike: Source, Counter-Strike: Condition Zero, Strike Suit Zero, Counter-Strike: Global Offensive, Bang Bang Racing, Pineapple Smash Crew, Commandos: Behind Enemy Lines, Catch a Falling Star, Sherlock Holmes: The Mystery of the Mummy, Mimpi.* This result was expected since, as we can see, all the Counter-Strike series games show up as well as some other shooter type games.

4.4 Evaluation

For evaluating this milestone's improvements, we chose to evaluate two of the queries from the last milestone against the new system, which do not trigger the "gamer mode". Its results should simply reflect the slight adjustments made to the *bf* and the existence of the new column for storing the entities' descriptions.
Later, we make two new queries that trigger the "gamer mode" to analyze this stricter system behaviour and its trade-offs.

(1) Which are the best games based on Vietnam?

```
Query manually curated in M2's style:
ResponseName:"Vietnam"~2^4,
PromotionalDescription:"Vietnam"~2,
WikiData:"Vietnam"~2^3
q.op= OR
fq= [Metacritic: [50 TO 100]]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))

Query generated by M3's CLI:
q= vietnam
q.op= OR
qf= ResponseName^4 Genres^3
WikiData^2 PromotionalDescription
EntityWikiData
bf= div(sum(sum(mul(0.55,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount))),5)
```

Table 13: Query 1 - Information retrieval effectiveness measures.

Metric	M2 System	M3 System
Average Precision (AvP)	0.64	0.82
Precision at 10 (P@10)	0.60	0.60
Recall at 10 (R@10)	1	1
F1-Score at 10 (F@10)	0.75	0.75

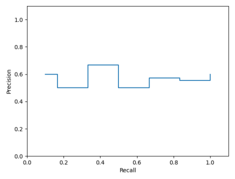


Figure 22: Query 1 - Precision-Recall Curve for M2 system.

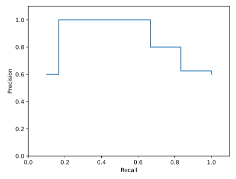


Figure 23: Query 1 - Precision-Recall Curve for M3 system.

Looking at table 13 and also figures 22 and 23, we can notice an improvement from the M2 system to the M3 system.

(2) Which are the best games to solve riddles?

```
Query manually curated in M2's style:
ResponseName:"riddles"~2^4
PromotionalDescription:"riddles"~2
wikiData:"riddles"~2^3
q.op= OR
fq = [Metacritic: [50 TO 100]]
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))

Query generated by M3's CLI:
q= riddles
q.op= OR
qf= ResponseName^4 Genres^3
WikiData^2 PromotionalDescription
EntityWikiData
bf= div(sum(sum(mul(0.55,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount))),5)
```

Table 14: Query 2 - Information retrieval effectiveness measures.

Metric	M2 System	M3 System
Average Precision (AvP)	0.64	0.57
Precision at 10 (P@10)	0.50	0.70
Recall at 10 (R@10)	0.71	0.88
F1-Score at 10 (F@10)	0.59	0.78

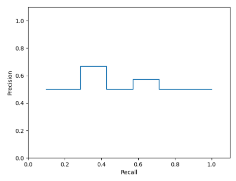


Figure 24: Query 2 - Precision-Recall Curve for M2 system.

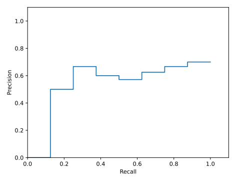


Figure 25: Query 2 - Precision-Recall Curve for M3 system.

Looking at table 14 and figures 24 and 25, we can recognize a slight improvement from the M2 system to the M3 system. In figure 25, we can see that the initial precision is 0. This happened because the first result returned by the M3's system includes in its *EntityWikiData* a person that worked on puzzle games, which is a synonym of riddles, even though that had nothing to do with the returned game, which is a quite popular one and got up there due to the *bf*.

(3) Which are the best open-world shooter games?

```
Query manually curated in M2's style:
q= open world shooter
q.op= OR
qf= ResponseName^4 Genres^3
WikiData^2 PromotionalDescription
EntityWikiData
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))

Query generated by M3's CLI:
q= open world shooter
q.op= AND
qf= ResponseName^4 Genres^3
WikiData^2 PromotionalDescription
EntityWikiData
bf= div(sum(sum(mul(0.85,Metacritic),
mul(4,log(SteamSpyPlayersEstimate))),
mul(21,log(RecommendationCount))),5)
```

Table 15: Query 3 - Information retrieval effectiveness measures.

Metric	M2 System	M3 System
Average Precision (AvP)	0.51	0.83
Precision at 10 (P@10)	0.60	0.60
Recall at 10 (R@10)	0.75	0.75
F1-Score at 10 (F@10)	0.67	0.67

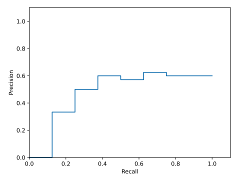


Figure 26: Query 3 - Precision-Recall Curve for M2 system.

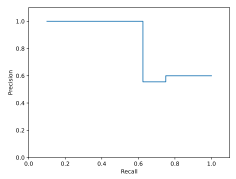


Figure 27: Query 3 - Precision-Recall Curve for M3 system.

Looking at table 15 and figures 26 and 27, we can recognize a noticeable improvement from the M2 system to the M3 system. In this case, the use of the AND operator, forced due to the "gamer mode" trigger, gives the M3 system a substantial advantage to find less popular but demanded by the experienced player games.

(4) Which are the best magic PvP games?

```
Query manually curated in M2's style:
q= magic pvp
q.op= OR
qf= ResponseName^4 Genres^3
WikiData^2 PromotionalDescription
EntityWikiData
bf= sum(sum(mul(0.5,Metacritic),
mul(10,log(SteamSpyPlayersEstimate))),
mul(15,log(RecommendationCount)))

Query generated by M3's CLI:
q= magic pvp
q.op= AND
qf= ResponseName^4 Genres^3
WikiData^2 PromotionalDescription
EntityWikiData
bf= div(sum(sum(mul(0.799,Metacritic),
mul(4,log(SteamSpyPlayersEstimate))),
mul(21,log(RecommendationCount))),5)
```

Table 16: Query 4 - Information retrieval effectiveness measures.

Metric	M2 System	M3 System
Average Precision (AvP)	0.68	0.74
Precision at 10 (P@10)	0.40	0.60
Recall at 10 (R@10)	0.44	0.67
F1-Score at 10 (F@10)	0.42	0.63

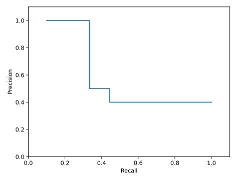


Figure 28: Query 4 - Precision-Recall Curve for M2 System.

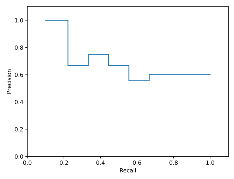


Figure 29: Query 4 - Precision-Recall Curve for M3 System.

The last evaluated query also used the new gamer mode in the M3 system, but this time only a slight improvement can be found after analysing table 16 and also figures 28 and 29.

Overall, the results are in line to what we expected.
When the "gamer mode" is not triggered, the new system usually returns more relevant first results, given it has more data at index time and a more sensible *bf* at query time. The rare cases when the searched term casually appears in an entities' description and it adds nothing to the domain information, as it is the case in query 2, are something to keep in mind, but are an acceptable trade-off.
When it is triggered, the behaviour is "riskier" in the way that popular games are often omitted due to them not matching exactly all terms of the query. We kept the need for precise matching in mind when constructing the *qrels* for these queries, so the results are better for the new system. An user expecting more generic results would be disappointed by this, though. This could perhaps be solved by mixing results from a query using OR and another using AND as the operators, but this is surely subjective ground.
The mean average precision for all the tested queries is coherent with the individual results.

```
Mean Average Precision (M2 System): 0.6175
Mean Average Precision (M3 System): 0.740
```

5 CONCLUSION

After cleaning up the dataset and extracting the most relevant features, our data processing pipeline returns a robust set of documents with plenty of text and a set of concepts that allowed the search engine to flourish and return pertinent results.
The last efforts aimed at developing a friendly user-facing interface to the search system, able to generate queries more aligned to the user's needs, proved to be interesting, usually returning more relevant results first, but skipping some big names in the dataset due to the less importance given to popularity.
Summing up, we are satisfied with the way the developed search system turned out to be. It returns relevant results based on plenty of textual data and is smart enough to modify its behavior to accommodate to the user's needs.
There is, for sure, some room for improvement, still. Gathering information about the user's habits and product preferences, as well as their feedback to previous query results, is a sensible step to take for someone looking to improve on this system's overall intelligence.

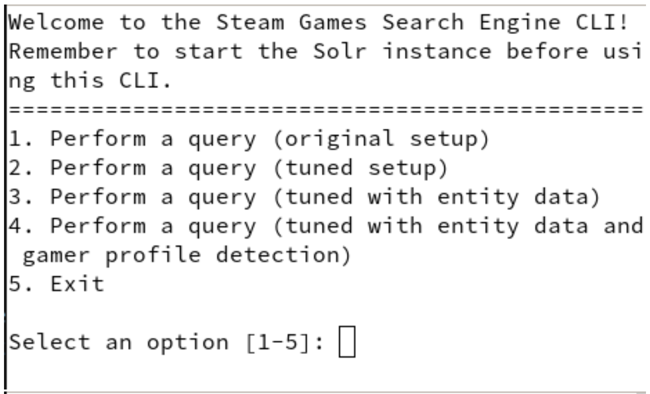
REFERENCES

[1] Kevin Bougé. 2011. Download synonyms. `https://sites.google.com/site/kevinbouge/synonyms-lists` Last accessed date: 14-10-2022.
[2] Valve Corporation. 2022. Steam: About. `https://store.steampowered.com/about/` Last accessed date: 12-10-2022.
[3] The Apache Software Foundation. 2022. Apache Solr. `https://solr.apache.org/` Last accessed date: 12-10-2022.
[4] The Apache Software Foundation. 2022. MoreLikeThis. `https://solr.apache.org/guide/8_10/morelikethis.html` Last accessed date: 15-12-2022.
[5] Wikimedia Foundation. 2022. Wikipedia. `https://www.wikipedia.org/` Last accessed date: 12-10-2022.
[6] Wikimedia Foundation. 2022. Wikipedia API. `https://steamcommunity.com/dev` Last accessed date: 12-10-2022.
[7] Game Genius. 2022. https://genius.com/Game-genius-list-of-gaming-terms-annotated. `https://genius.com/Game-genius-list-of-gaming-terms-annotated` Last accessed date: 15-12-2022.
[8] Matthew Honnibal. 2022. spaCy. `https://spacy.io/` Last accessed date: 15-12-2022.
[9] Bryan Horling and Matthew Kulick. 2009. Personalized Search for everyone. `https://googleblog.blogspot.com/2009/12/personalized-search-for-everyone.html` Last accessed date: 15-12-2022.
[10] Craig Kelly. 2016. Steam Game Data. `https://data.world/craigkelly/steam-game-data` Last accessed date: 12-10-2022.
[11] Sérgio Nunes. 2021. Search Results Evaluation. `https://git.fe.up.pt/pri/tutorials/-/tree/main/06-evaluation` Last accessed date: 16-10-2022.
[12] Matt Southern. 2020. Over 25% of People Click the First Google Search Result. `https://www.searchenginejournal.com/google-first-page-clicks/374516/` Last accessed date: 15-10-2022.
[13] Steam Spy. 2021. Steam Spy: About. `https://steamspy.com/about` Last accessed date: 12-10-2022.
[14] Steam. 2022. Steam API. `https://steamcommunity.com/dev` Last accessed date: 12-10-2022.
[15] Data World. 2016. Data World. `https://data.world` Last accessed date: 12-10-2022.

A SCHEMA ADDITIONS

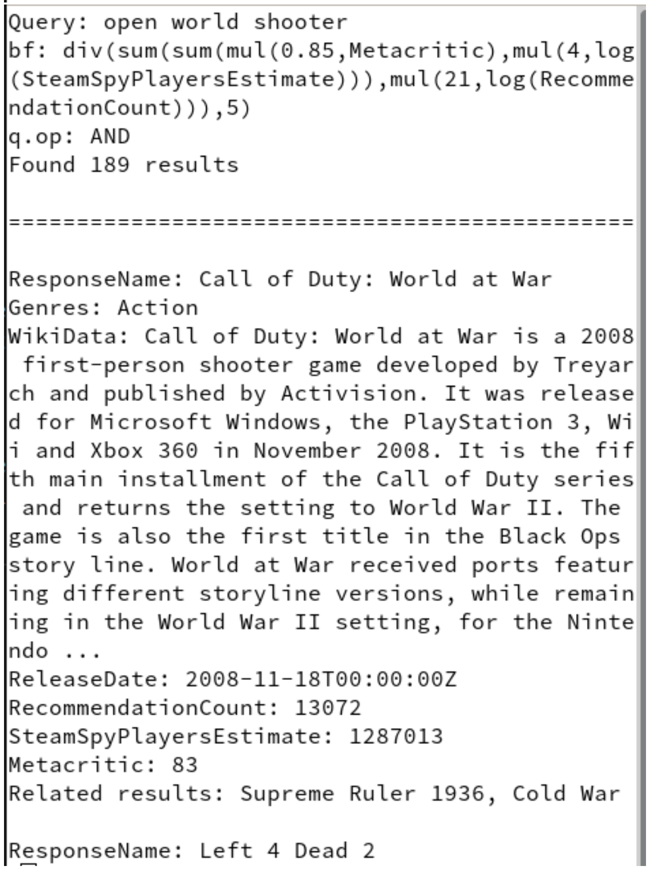
```
{
  "add-field-type": [
    {
      "name": "mdate",
      "class": "solr.DateRangeField",
      "multiValued": false
    },
    {
      "name": "mbool",
      "class": "solr.BoolField",
      "multiValued": false,
      "docValues": true
    },
    {
      "name": "mint",
      "class": "solr.IntPointField",
      "multiValued": false,
      "docValues": true
    },
    {
      "name": "mfloat",
      "class": "solr.FloatPointField",
      "multiValued": false,
      "docValues": true
    },
    {
      "name": "mstring",
      "class": "solr.StrField",
      "multiValued": false
    },
    {
      "name": "short_text",
      "class": "solr.TextField",
      "indexAnalyzer": {
        "charFilter": {
          "class": "solr.MappingCharFilterFactory",
          "mapping": "mapping-FoldToASCII.txt"
        },
        "tokenizer": {
          "class": "solr.ClassicTokenizerFactory"
        },
        "filters": [
          {
            "class": "solr.ClassicFilterFactory"
          },
          {
            "class": "solr.KStemFilterFactory"
          },
          {
            "class": "solr.PhoneticFilterFactory",
            "encoder": "DoubleMetaphone"
          },
          {
            "class": "solr.LowerCaseFilterFactory"
          }
        ]
      },
      "queryAnalyzer": {
        "charFilter": {
          "class": "solr.MappingCharFilterFactory",
          "mapping": "mapping-FoldToASCII.txt"
        },
        "tokenizer": {
          "class": "solr.ClassicTokenizerFactory"
        },
        "filters": [
          {
            "class": "solr.ClassicFilterFactory"
          },
          {
            "class": "solr.KStemFilterFactory"
          },
          {
            "class": "solr.LowerCaseFilterFactory"
          }
        ]
      }
    },
    {
      "name": "general_text",
      "class": "solr.TextField",
      "indexAnalyzer": {
        "charFilter": {
          "class": "solr.MappingCharFilterFactory",
          "mapping": "mapping-FoldToASCII.txt"
        },
        "tokenizer": {
          "class": "solr.StandardTokenizerFactory"
        },
        "filters": [
          {
            "class": "solr.LowerCaseFilterFactory"
          },
          {
            "class": "solr.CommonGramsFilterFactory"
          },
          {
            "class": "solr.SynonymFilterFactory",
            "synonyms": "synonyms_en.txt",
            "ignoreCase": true,
            "expand": true
          },
          {
            "class": "solr.StopFilterFactory",
            "ignoreCase": true,
            "words": "lang/stopwords_en.txt"
          },
          {
            "class": "solr.RemoveDuplicatesTokenFilterFactory"
          },
          {
            "class": "solr.PorterStemFilterFactory"
          }
        ]
      },
      "queryAnalyzer": {
        "charFilter": {
          "class": "solr.MappingCharFilterFactory",
          "mapping": "mapping-FoldToASCII.txt"
        },
        "tokenizer": {
          "class": "solr.StandardTokenizerFactory"
        },
        "filters": [
          {
            "class": "solr.LowerCaseFilterFactory"
          },
          {
            "class": "solr.StopFilterFactory",
            "ignoreCase": true,
            "words": "lang/stopwords_en.txt"
          },
          {
            "class": "solr.CommonGramsFilterFactory"
          },
          {
            "class": "solr.PorterStemFilterFactory"
          }
        ]
      }
    }
  ],
  "add-field": [
    {
      "name": "ResponseName",
      "type": "short_text"
    },
    {
      "name": "PromotionalDescription",
      "type": "general_text"
    },
    {
      "name": "WikiData",
      "type": "general_text"
    },
    {
      "name": "EntityWikiData",
      "type": "general_text"
    },
    {
      "name": "ResponseID",
      "type": "mint"
    },
    {
      "name": "ReleaseDate",
      "type": "mdate"
    },
    {
      "name": "RequiredAge",
      "type": "mint"
    },
    {
      "name": "DemoCount",
      "type": "mint"
    },
    {
      "name": "DeveloperCount",
      "type": "mint"
    },
    {
      "name": "DLCCount",
      "type": "mint"
    },
    {
      "name": "Metacritic",
      "type": "mint"
    },
    {
      "name": "RecommendationCount",
      "type": "mint"
    },
    {
      "name": "PublisherCount",
      "type": "mint"
    },
    {
      "name": "SteamSpyOwners",
      "type": "mint"
    },
    {
      "name": "SteamSpyPlayersEstimate",
      "type": "mint"
    },
    {
      "name": "AchievementCount",
      "type": "mint"
    },
    {
      "name": "ControllerSupport",
      "type": "mbool"
    },
    {
      "name": "IsFree",
      "type": "mbool"
    },
    {
      "name": "FreeVerAvail",
      "type": "mbool"
    },
    {
      "name": "PurchaseAvail",
      "type": "mbool"
    },
    {
      "name": "SubscriptionAvail",
      "type": "mbool"
    },
    {
      "name": "PriceInitial",
      "type": "mfloat"
    },
    {
      "name": "PriceFinal",
      "type": "mfloat"
    },
    {
      "name": "HeaderImage",
      "type": "mstring"
    },
    {
      "name": "PCMinReqsText",
      "type": "short_text"
    },
    {
      "name": "LinuxMinReqsText",
      "type": "short_text"
    },
    {
      "name": "MacMinReqsText",
      "type": "short_text"
    }
  ]
}
```

B CLI SCREENSHOTS



```
Welcome to the Steam Games Search Engine CLI!
Remember to start the Solr instance before usi
ng this CLI.
=============================================
1. Perform a query (original setup)
2. Perform a query (tuned setup)
3. Perform a query (tuned with entity data)
4. Perform a query (tuned with entity data and
 gamer profile detection)
5. Exit

Select an option [1-5]:
```



```
Query: open world shooter
bf: div(sum(sum(mul(0.85,Metacritic),mul(4,log
(SteamSpyPlayersEstimate))),mul(21,log(Recomme
ndationCount))),5)
q.op: AND
Found 189 results

=============================================


ResponseName: Call of Duty: World at War
Genres: Action
WikiData: Call of Duty: World at War is a 2008
 first-person shooter game developed by Treyar
ch and published by Activision. It was release
d for Microsoft Windows, the PlayStation 3, Wi
i and Xbox 360 in November 2008. It is the fif
th main installment of the Call of Duty series
 and returns the setting to World War II. The
game is also the first title in the Black Ops
story line. World at War received ports featur
ing different storyline versions, while remain
ing in the World War II setting, for the Ninte
ndo ...
ReleaseDate: 2008-11-18T00:00:00Z
RecommendationCount: 13072
SteamSpyPlayersEstimate: 1287013
Metacritic: 83
Related results: Supreme Ruler 1936, Cold War

ResponseName: Left 4 Dead 2
```