Optional semester project - Autumn 2015

Swapi - A swapping recommender system

Johan Droz

Table of contents

Introduction

In the past, trade used to be the only system by which goods and services were exchanged, but nowadays its popularity decreased with the rise of modern economics [2]. It is nevertheless still used: some dedicated websites such as BookMooch [4] and the Gameswap subreddit [3] allow their members to exchange respectively books and video games. The Internet and social networks allow to connect an important number of people without geographic constraints and therefore expose the goods and services to a wider community increasing the chances to find trading partners.

In order to make an exchange on these websites, a user has to scroll through the proposals and when he finds something interesting, he has to contact the person to propose a trade.

Swapi is a project that consists in developing a framework providing recommendations of exchange.

Research directions

As Swapi is a research project, it can happen that the objectives change to match the current research direction. In this case, three directions were taken.

1. The first goal was to build Swapi its own website, to which users from all over the world can connect and exchange objects and services. Meanwhile, we were in contact with some administrators of the Gameswap subreddit to integrate Swapi in their platform.
2. Upon receiving a promising answer, we put the website aside and switched to build a bot that can interact with reddit's users and send them suggestions.
3. And finally after not hearing from reddit, the objective changed a last time: A real world dataset was extracted from the Gameswap subreddit in order to validate the recommender system.

A more detailed description is available in the next sections.

Swapit platform

General description

The first goal is to enhance the prototype of the Swapi website with new functionalities, namely every user should be able to log in with different types of accounts such as Facebook, Twitter and Google. The idea behind it is to attract more users to our platform and encourage them to share content on social networks, i.e. tweet about an interesting proposal, post some exchange proposal on Facebook...

Details

The existing prototype is a play-java application using Play 2.3.8 (java version), Scala 2.11 and SecureSocial master-SNAPSHOT. Users can connect to the website and create posts with items they want to trade. The initial goal is to allow authentication support for leading services like Google, Facebook, Twitter and possibly other services.

Facebook and Google are using OAuth2, while Twitter uses OAuth1. This difference is insignificant because SecureSocial provides off the shelf support for all of them.

Status

The authentication works for all of the services separately, but there is a problem when the users select which service they want to use.

The website was suddenly abandoned to integrate Swapi in Gameswap, so this bug was not solved.

Gameswap subreddit

Gameswap is a subreddit, in other world a subpart of reddit dedicated to the exchange of video games. Here, a large community of users can interact and trade all kinds of video games for any existing console.

When a user wants to trade a video game, he simply writes a text post containing a title, a body and a comment section. The title should respect the format in Figure 1, where H stands for have and W for want. In the body, specifics about the swap are described. Finally, a comment section allows interested people to make contact, ask questions and negotiate a trade.

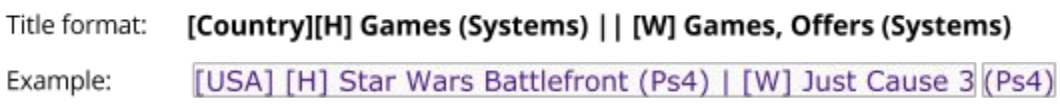

Figure 1: Gameswap title format

Usually the negotiation take place here, but it happens that users switch to private messages to carry on the discussion. Once both agree on a deal, one of the user create a post in a dedicated thread called “Completed Trade Thread”. This confirmation post includes the following information [3]:

- The games that were swapped
- The swapping partner’s name
- A confirmation from the partner
- A link to the comment in the trade thread where negotiations took place

Integrate Swapi in the Gameswap subreddit

General description

In order to validate the recommender system, we wanted to deploy it in a real-world application, namely the “Gameswap” section of the reddit website.

The major limitation of the exchange process used by Gamswap is that users have to parse the proposals to find an interesting deal.

Our goal is to use the Swapi recommender system to make users’ life easier by contacting them directly and suggesting trades that may be interesting for them.

Details

Swapi suggests a trade to users by sending an automatic message like the one in Figure 2. To achieve this, Swapi must be able to interact with the Gameswap subreddit. It needs to access new posts to see users wishes and possessions as well as write comments to propose trades to users.

We chose to create a reddit bot using jReddit, a Java wrapper for Reddit API. Among other things, jReddit can login with a user, retrieve information, make comment on submissions, send and receive messages and notifications [7].


Hi human,

I'm Swapi, the bot of gameswap.

I make recommendations to subscribers of gameswap according to their preferences. I can send you PM's when interesting items appear. For example, I think you might be interested in this user submission, why not swapping "X" for "Y" ?

If you want such recommendations, just subscribe.

Figure 2: Swapi recommendation message

Status

For now, the bot is able to login with a username, which is temporary. We are waiting for the Gameswap administrator's approval to integrate it into the subreddit. The bot can retrieve submissions on Gameswap, process and store them.

There is still the need to improve the system sorting the games. An interface with the recommender system should also be built, to exchange data in a format that Swapi can handle, namely we should work with identification numbers. Each game and user should have it's own unique id.

For now, the extracted data are stored in a SQL database. But there is a problem with the game names: as they are extracted from the posts, two same games can have different identification numbers. Indeed let's imagine two users, one wanting "Assassin's creed IV" and the other "Assassin's creed: Black Flag". In reality those are the same game, but in the database they are considered different. This is a considerable issue because the recommender system will not be able to work with a wrong dataset and its performance will greatly suffer.

Upon not receiving answer from the administrators of Gameswap, the project of integrating Swapi was abandoned.

Building a testing dataset

General description

In order to test the recommender system, a dataset is needed. Here we chose to extract a real-world dataset from bartering online platforms, namely the Gameswap subreddit.

To make the dataset usable, different kind of information have to be extracted. Ideally, for each user we want:

- his itemlist or give-away list (the games he owns and wants to trade).
- his wish list (the games he wants).

Additionally, we need to know the trades that actually happened.

The wish list and itemlist of each user are extracted from the title and body of the Gameswap's posts and the successful trades can be extracted from the "Completed Trade Thread" of Gameswap.

Details

In theory, extracting the data from Gameswap is an easy task. Indeed, when a user writes a post to ask for an exchange, he is asked to follow the format of Figure 1. With the correct format, a simple regular expression is able to extract all information needed.

But unfortunately, the format is not enforced and users are allowed to post anything. In practice, we get everything from post deviating slightly from the expected format to texts that are utterly different.

Game written by users	Official name
tloz:albw	The Legend of Zelda: A Link Between Worlds
dw8xlce	Dynasty Warriors 8: Xtreme Legends Complete Edition
sotcico	The Ico & Shadow of the Colossus Collection

Table 1: Examples of acronyms

To test swapi, we need an accurate list of games, and this create lots of difficulties:

- Same game with different name: e.g. Assassin's Creed IV, Assassin's Creed 4 and Assassin's Creed: Black Flag refer to the same game
- The acronyms: it is really common for users not to write the complete game name but to use acronyms instead. Table 1 shows such examples.

To that, we add all the grammatical and spelling mistakes as well as the typo made by users, and it become extremely tricky to build a proper dataset.

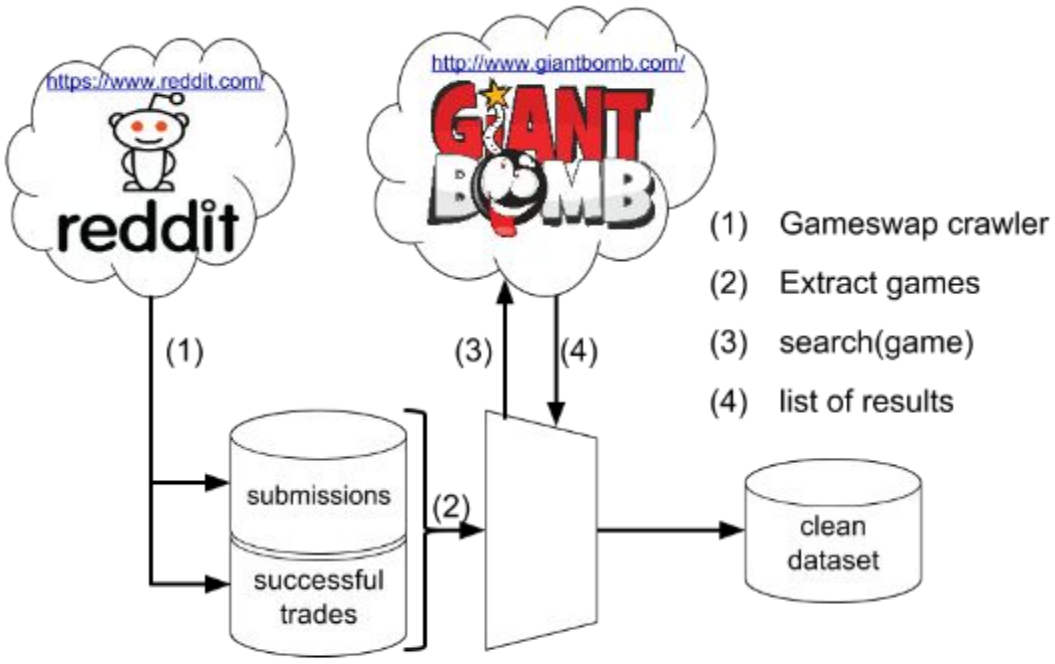

Figure 3: Building the dataset

We already have the data “submissions” and “successful trades”. The dataset is built in two phases, as depicted in Figure 3. First we extract potential games names from a post (2), then these names are validated (3), (4). We need to validate the names for two reasons:

1. As there is no precised structure in the posts, sometimes we mistakenly take arbitrary words for a game name.
2. Users are referring to the same game with different names (e.g. “Assassin’s creed IV”, “Assassin’s creed 4” and “Assassin’s creed: Black Flag”)

The validation process allows to discard names in the first case, and deal with different naming and spelling variations of the same game in the second.

Description of the data

A crawler has extracted data from Gameswap and saved it to two JSON files: “submissions” and “success_thread” (See Figure 3). The former contains the posts and associated comments and the latter contains the successful trades.

In the “submissions” file, 14890 different users made 41086 posts. In these posts, they negotiate exchanges via the comment section. 666936 comments are included in the file.

In “success_thread”, there are 6634 posts each representing a successful swap of one or more games between two users. However, only a fraction of it can be exploited to build the final dataset. Indeed, in more than 2200 posts, that is ⅓ of the entire thread, there is no swap but only a user paying money to another for a game. Now if we count the exchanges with a confirmation of both users, we are down to 3500 posts.

Extracting game names

To build the dataset, each post is processed along with its associated comments. A wish list and itemlist are extracted from the title and the body of the posts.

The title is supposed to have an imposed format (Figure 1). The list of games a user owns follows the tag “[H]” and the games he wants follows “[W]”. In practice, this is not strictly

respected, but some variations happened. The tags are not always those two, but can change to [have], (h) , [want] or (w) for example. So a list of these tags was extracted from the dataset to allow us to match them and extract the associated lists. Note that the posts without tags are dropped.

For the body, the task is more complicated because there is no predefined format, so users just write specifications in an english that is not always accurate. To extract game names we look for specific words preceding the list, e.g. “want”, “looking for” for the wish list or “have”, “for trade”, “got” for the itemlist. Then we assume that what follows is a list of games and extract it. It may not always be the case, hence a validation process follows.

Here again if there are no structure, the posts are dropped.

Validating game names

To validate a game name, we query a videogame website with the extracted name and try to find a match in the answers. The idea here is that if we find a match, we are indeed dealing with a videogame, while in the opposite case there may be a mistake with the extraction of the name.

We chose the GiantBomb website [6] because it has more than 48’000 games for all platforms in its database and a python wrapper for the GiantBomb API is provided. It allows us to search for video games directly from a python script.

The validation process is depicted in Figure 3. GiantBomb is queried with all games extracted from the JSON files. We receive in return a list containing the 5 games names that match best the query, then we find the best correspondence in this list and if the match is close enough to the initial query we include the game in the final dataset. Otherwise the information is dropped.

To compare two names, we use the Levenshtein distance, or edit distance, which is defined as the minimum of single character edit required to change one name into the other [1]. If this distance is below a threshold, the string extracted from the post is indeed a game name.

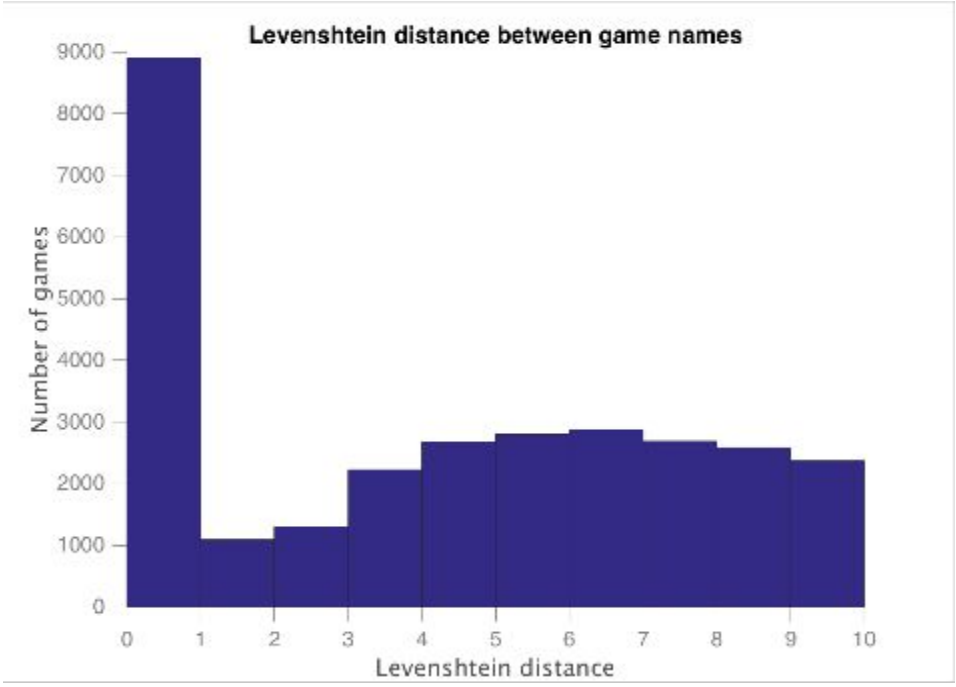

Figure 4: Histogram of the Levenshtein distance between the extracted game and answer from GiantBomb

Setting a good threshold is not simple. If it is set to 0, only exact matches will end up in our final dataset, but lots of valid games will be discarded because of insignificant differences with the name returned by GiantBomb, such as a punctuation difference, or a small typo. On the other hand, if this threshold is too big, wrong games will be in the final dataset which is not acceptable to validate the recommender system. After experimenting, the threshold was fixed to 1. Figure 4 shows the histogram of the Levenshtein distance between all extracted game names and GiantBomb answers. We can see that there are about 9000 exact matches. The analyse of the games with a distance of 1 determine than they are discarded because of a spelling mistakes, typo or punctuation difference so we can add them to the final dataset too. It is the same story when the distance is 2, but this time there are wrong games too, e.g. the query “megaman” match “Megamania” but the real game should be “Mega Man”. So the threshold is fixed to 1.

After analysing the games rejected, it appears that often the games are discarded because of the following reasons:

- Punctuation differences (e.g. “Mario Kart Double Dash” and “Mario Kart: Double Dash!!” have a distance of 3)
- Information added by users (e.g. “Super Mario for DS” and “Super Mario” have a distance of 7)
- Acronym are often used on Gameswap (e.g. “ff14” and “Final Fantasy XIV” have an edit distance of 15)
- One use integer, the other roman numerals (e.g. “Final Fantasy 14” and “Final Fantasy XIV” have a Levenshtein distance of 3)

The scripts created to build the dataset take all these cases into account. Each time the best match is kept. When numbers are detected, the distance is computed twice. For the second time the number are converted into roman numerals. A dictionary with more than 1100 acronyms related from gaming taken from [5] was build. If an abbreviation is found in a game name we try to replace it. The punctuation is removed and finally if a console name appears in a name, we try to remove it. Table 2 quantifies the games matched by the different rules. Note that the rules are cumulative, i.e. in the line “Replace acronyms” the two precedent ones are active too.

Filter	# games in wishlist	# games in itemlist
No rules (only Levensthein <= treshold)	18’676	17’069
Remove punctuation	19’032	17’479
Remove some keywords (useless info added by users)	20’244	18’363
Replace acronyms	20’954	19’245
Replace numerals by roman numbers	21’946	20’338

Table 2: Summary of the number of games validated with the different rules

All those rules work surprisingly well: the script is able to match games such as “cod:mw3” with “Call of Duty: Modern Warfare 3” and “ff 13-2” with “Final Fantasy XIII-2” which otherwise would have been rejected for having huge Levenshtein distance. A bit more than 6800 games that were initially rejected can be added to the final dataset due to these rules.

Success Trade Thread

To cross validate the recommender system, the dataset needs to contain successful trade between users.

More precisely, we need a file containing the details of each swap. This file must contain an entry per swap, with both IDs of the trading partners and IDs of the games that are exchanged.

We saw in the section “Description of data” that there are only about 3500 exploitable posts. But in reality, this number decreases again. A more detailed analysis of the posts shows that users are surprisingly often exchanging goods other than video games. They exchange anything from object related to gaming like consoles, controllers, memory cards to other collectable such as comic books, figurines and board games… As we are only interested in the exchange of games, those entries are discarded and we are left with only 1800 usable posts which is roughly 27% of the initial thread. Table 3 summarizes the different filtering steps.

When using the same technique (name extraction and validation), we are able to come up with only 242 entries which is clearly not enough to validate the recommender system. This is due to the lack of structure in the comments, the “forum like” type of writing, by that I mean the extensive use of abbreviations, grammatical mistakes, typos,… Moreover it happens relatively often that only a part of the trade was successfully validated which make it useless too.

The script can be improved, but it will be extremely difficult to reach a sufficient number of trades (initially we hoped to have at least 1000 entries) while keeping a good accuracy. Due to the lack of time and the urgent need of this dataset, the list of successful trade was manually sorted. This time we obtain 1794 successful swap. We cannot do better with this dataset.

Filter	# games discarded	# games left
No filter	0	6634
Discard money transfer	2206	4428
Discard unconfirmed trades	933	3495
Manual sort	1701	1794

Table 3: Summary of the filtering steps for the success trades

Limitations of the validation method

Because of the lack of structure, lots of game are discarded from the final dataset. Even though the methods previously described work well, they are limited. For example the trick of replacing acronyms with the full name works well, but not all acronyms are in the dictionary and sometimes there is ambiguities: the same abbreviation changes depending on the context, e.g. “ac” can mean either “Assassin’s Creed” or “Animal Crossing”, “mk” is either “Mario Kart” or “Mortal Kombat”.

By setting the Levenshtein threshold to 1, we allow users to make 1 typo or spelling mistake and still add the game in the final dataset. If there is more difference, the game is discarded.

We directly depend on the GiantBomb search engine. Indeed every time GiantBomb does not return the exact game mentioned by users in the list, this particular game will be discarded. Fortunately, in practice the results are good.

Testing final dataset

The final dataset is stored into 3 JSON files for the itemlist, the wish list and the successful trades. The number of elements in each file is reported in Table 4.

file name	Users	Games	Different games
wishlist	7'080	21'946	2'451
itemlist	7'308	20'338	2'385
success trades	1'395	4'898	1'240

Table 4: Final dataset

Those files contain only identifications number for games and users, so it is difficult to see if the information is coherent or not. A testing scripts was designed with this purpose. It parses all files, gets the real usernames and game names from the identification numbers, and checks that those information coincide with the original dataset.

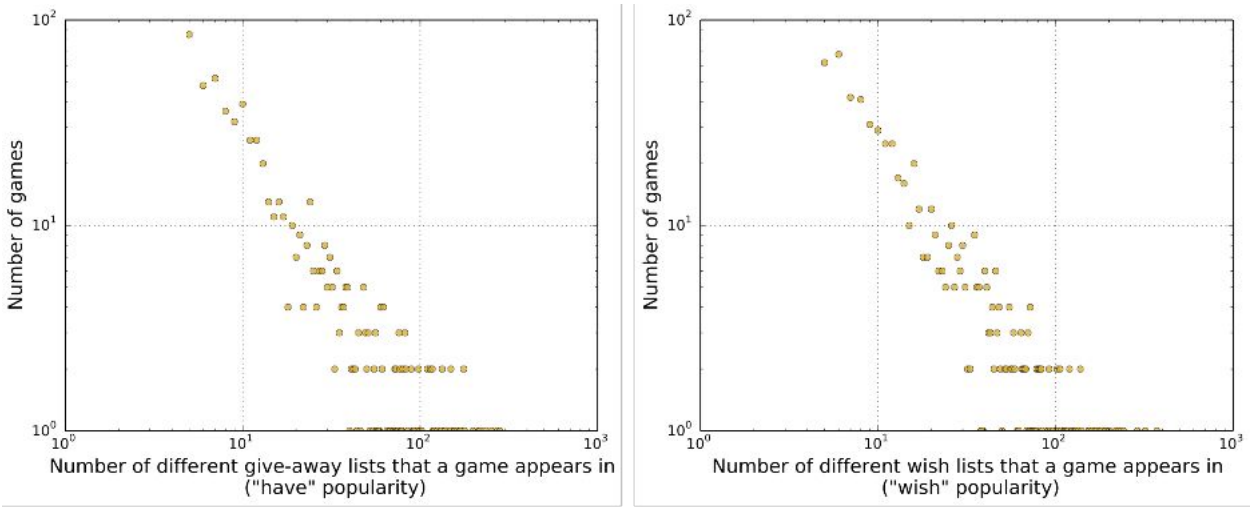

Figure 5: Plot of the popularity of the games in the itemlist and wishlist

Moreover, a few plots with the data of the itemlist and wish list have been done. Figure 5 shows the popularity of the games in the itemlist (left plot) and wish list (right plot). We can see that they follow a power law, which was the expected result. That means that games that appear in lot of lists are pretty rare, while game appearing in a small number of lists are more common.

Figure 6 shows the size of the itemlist (left plot) and wish list (right plot). As expected, they also follow a power law.

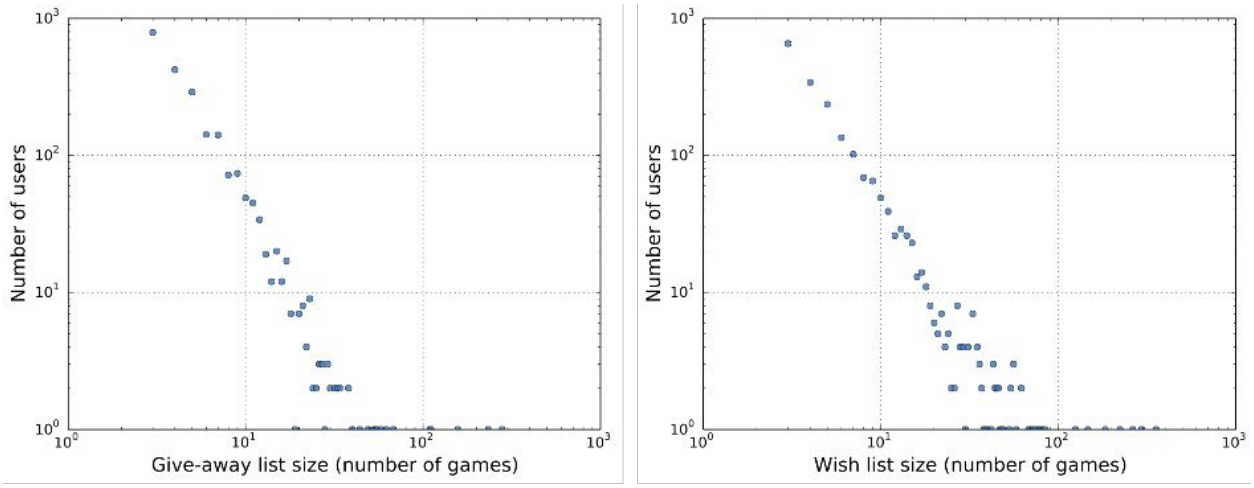

Figure 6: Plot of the size of the itemlist and wishlist

Conclusion

A clean dataset representing users possessions as well as their wishes was created from the Gameswap subreddit.

The time needed to build the dataset was underestimated. A first shallow analysis suggested that the task was supposed to be quite easy. But unfortunately, users not respecting the format increase the complexity of the work. In the case of the success trades, the inability to automatically create a dataset of sufficient size led to a manual sorting. This is a tedious but rewarding task because all relevant trades have been included in the final dataset. This cannot be equalled with the use of an automatic sorting.

However, the wish list and itemlist were extracted automatically. So many game were discarded because they did not match their official names. As improvements, we can think of increasing the size of both lists by either improving the process or using crowdsourcing.

Moreover, for now we are not taking into account the console type, in real life application swapi must be able to distinguish different gaming platforms. Indeed it is not enough to recommend the right game, but the platform has to be correct if a swap is to be made.

Bibliography

[1] “Levenshtein distance” *Wikipedia*. Wikimedia Foundation. Web. 09 Dec. 2015 <https://en.wikipedia.org/wiki/Levenshtein_distance>

[2] “Trade” *Wikipedia*. Wikimedia Foundation. Web. 03 Jan. 2016 <https://en.wikipedia.org/wiki/Trade#Prehistory>

[3] "/r/Gameswap - All Your Swaps Are Belong to US! • /r/gameswap." *Reddit*. Web. 23 Dec. 2015 <https://www.reddit.com/r/gameswap/>

[4] "BookMooch." *: Trade Your Books with Other People*. Web. 23 Dec. 2015 <http://bookmooch.com/>

[5] "Computing » Gaming." *Gaming Abbreviations*. Web. 03 Jan. 2016 <http://www.abbreviations.com/acronyms/GAMING>

[6] "Giant Bomb." *List of Video Games - Giant Bomb*. Web. 03 Jan. 2016 <http://www.giantbomb.com/games/>

[7] "JReddit." *Java wrapper for Reddit API*. Web. 23 Dec. 2015 <http://jreddit.github.io/>