Università di Pisa

Master Degree in Artificial Intelligence and Data Engineering

Business and Project Management

Videogames Market Search

Tommaso Amarante, Giacomo Pacini

Github repository:
https://github.com/Ruggero1912/bpm-videogames-market-search

Contents

Chapter 1

Introduction

The videogame industry is a constantly evolving area in which it is essential to listen to market feedback on products released and understand what the public wants to be able to create a successful product.

It is really important to determine what people want the most from a game in order to create a best selling product and not to waste energies in non relevant features.

1.1 Research question

We assumed the point of view of a software house that is planning to develop a new videogame of a certain genre and which wants to find what are the features on which it should invest and work the most.

1.2 Goal of the project

The aim of our project is to analyze the market feedback about popular games of a given genre in order to find the most relevant features that people talk about.

We will exploit a huge set of reviews to perform our analysis.

Chapter 2

Process

2.1 Dataset

First of all we chose a dataset of about 6 milions Steam reviews where each row is made of the name and the review, since that datset does not contain the genre we used the Steam API in order to gather all the genres.

```
class SteamAPI:
    STEAMAPIBASEURL = ”https://store.steampowered.com/api/-API ”
    STEAMAPIAPPDETAILSURL = STEAMAPIBASEURL.format(API=”
appdetails/”)

    WAITTIMER = 300  # 5 minutes

    def printapiurl():
        print(SteamAPI.STEAMAPIAPPDETAILSURL)

    def getappdetails(appid, language='english'):
        ”””
        receives as input a Steam appid and returns a dict containing all
the app details or null if the appid was not found
        ”””
        params = “-
            ”l” : language,
            ”appids” : appid
        “
        response = requests.get(SteamAPI.STEAMAPIAPPDETAILSURL,
params=params)

        if(response.statuscode == 429):
            print(”“n[-datetime ] You have been banned, going to lock the
script for -x  seconds :) After that time I will retry the request for
the appid '-appid '...”.format(x=SteamAPI.WAITTIMER, appid=appid,
 datetime=datetime.now()))
            time.sleep(SteamAPI.WAITTIMER)
            return SteamAPI.getappdetails(appid, language=language)


        if(response.text in [None, ”null”] or response.statuscode != 200):
```

```
            print(f"There was an error handling the request for the app id
   -appid , status code: -response.statuscode , response.text: -response.
   text ")
            return None
        dictresponse = json.loads(response.text)
        return dictresponse
```

Listing 2.1: our SteamAPI client class

Then we joined the list of genres found for each videogame with each review and we discarded the reviews of games without genre; at this point we have a complete dataset from which we can perform our main analysis.

2.2 Keyword Extraction Top Down Lexicon based approach

In the first place we tried to extract the keywords with two different approaches, first of all we filtered the reviews by a certain genre, in our case *Action*:

- **Extract the keywords from a unique text**: we tried to join each review in a unique string, this approach is not feasible due to the fact that the resulting string is too big to be stored in memory for the keyword extraction.

- **Extract the keyword from each review**: this approach was not optimal since the output would have been composed by a large number of keywords that would have been hard to analyse.

In order to reduce the amount of data to be processed, we decided to change point of view, adopting a **top down approach** for which we needed a dictionary of meaningful tags that will be searched inside the reviews. The reviews that do not contain at least one tag will be discarded.

In this way at the end of the process we will have a feature *tags* for each review in which there will be an array of tags that appear inside that review.

2.2.1 Building the dictionary

To build the dictionary we decided to exploit the IGDB API. IGDB is a service that collects information about every videogame released ever and exposes a free API useful to extract infos about the games.

In particular firstly we requested to the IGDB API, using the query syntax of IGDB API v4, the first 500 games of genre **action** ordered by number of press reviews.

```
fields *;
where keywords != null;
where totalratingcount != null;
sort totalratingcount desc;
limit: 500;
```

Listing 2.2: IGDB API v4 query to load most voted games

At this point we made another query to the IGDB API for each game of the list in order to load the tags ids list for the game.

Then we stored the number of times that each tag appeared for a game, and in the end we loaded from IGDB the slug of each tag.

This resulted in an array of meaningful tags. We counted the occurrences in order to count the weight for each tag and we discarded the ones with a weight count lower than 20.

This approach reduced the size of the dictionary of tags from 10000 tags to 929 tags.

2.2.2 Dictionary voting

Once the dictionary was made, we observed that some of the selected tags were not meaningful and so there was the need of another filtering phase, that we decided to implement manually, considering the low amount of data.

We adopted a voting strategy for which each tag of the dictionary had to be evaluated by the members of the team with a score from 0 to 2 (2.1).

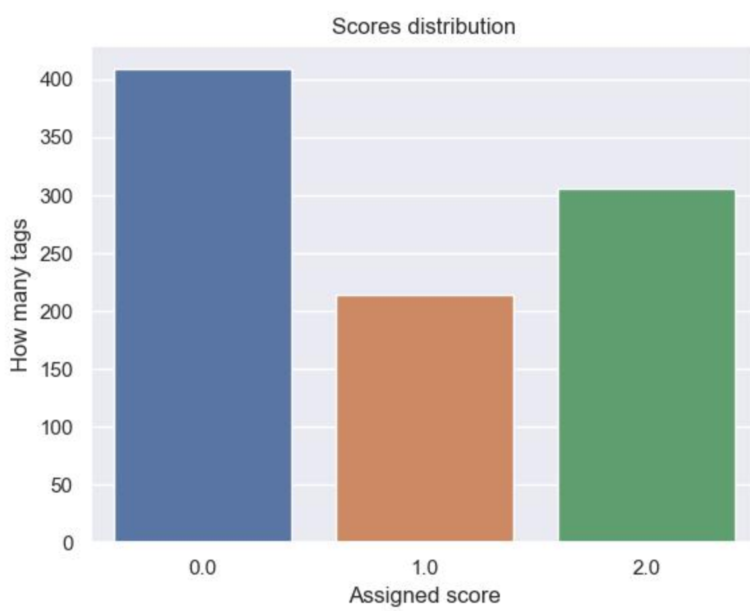

Figure 2.1: Tags scores distribution

We decided to keep only the tag words that totalized a score of 2/2 that are in total **306**.

2.2.3 Dictionary terms search inside reviews

After we have optimized dictionary with only meaningful tags for our aim, we performed a keyword search for each review in the original dataset, we then added a column that contains a list that contains the keywords found in the respective review.

We dropped the reviews that does not contain any of the tags so that our dataset is composed only of meaningful reviews.

At this point, we can choose a particular genre in order to obtain a general overview. Moreover, we did our analysis using the tags that have 2 as score.

After that process we obtained a list of keywords that are contained in the reviews, we manipulated the list in order to transform the plural keywords into singular ones, we counted the occurrences and we obtained a dataframe with all the keywords and the relative weight represented by the number of time that a certain keyword appeared.

2.3 Word cloud

The first analysis that we have performed is a visual representation of our keywords, in fact we decided to exploit the word cloud. In this representation the more the word is important the bigger it is into the image. In our case the more the keyword is recurrent the more is important. The result is shown in image 2.2.

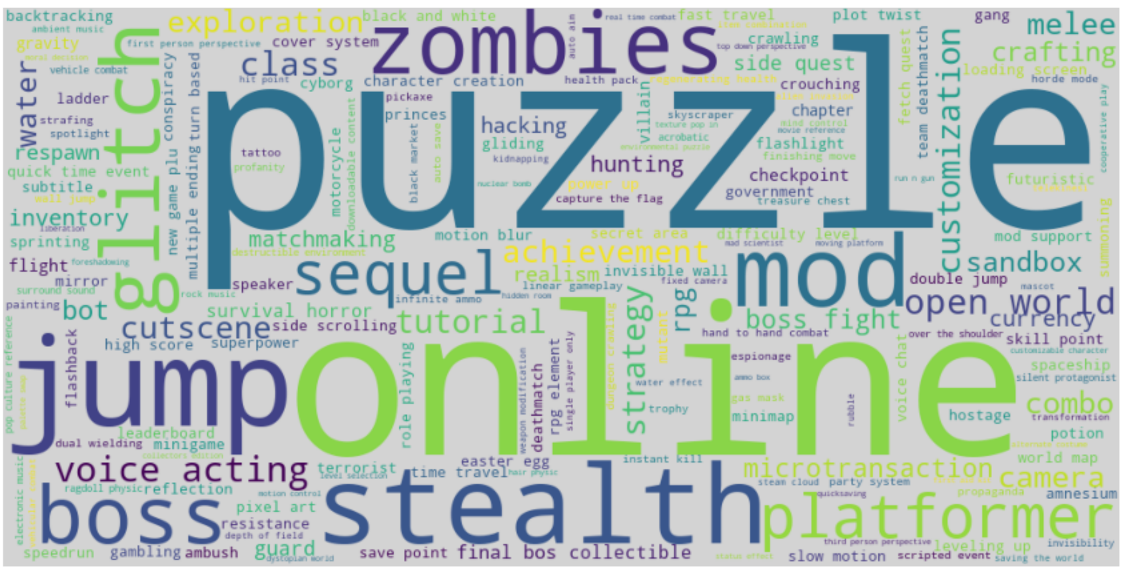

Figure 2.2: WordCloud

With this representation a software house can easily have an overview on the features that are fundamental for people's opinion.

Even though we did not implemented a sentiment analysis, for a software house it would be clear that 'glitch' has a negative connotation while, for example, 'mod' compatibility is a quite important feature for the videogame.

2.4 Words relations graph

To make a deeper analysis we built a graph in which each node is a keyword and each edge is drawn if two keywords appear in the same review. Each element in the graph has a weight, in particular the weight of each node represents how frequent a keyword appears in general, the weight of each edge represents how frequent two keywords appears together.

In order to make the graph clear enough to be human readable we dropped the isolated nodes and the edges with a weight lower that a certain threshold. We did this because if the edge's weight is lower than the threshold the edge can be considered not relevant for our analysis.

We also normalized all the weights using the maximum and the minimum values in order to make the graph human readable, as shown in figure 2.3.

This approach can be useful because a software house can visualize easily which couple of feature should be developed together to produce a best selling product.

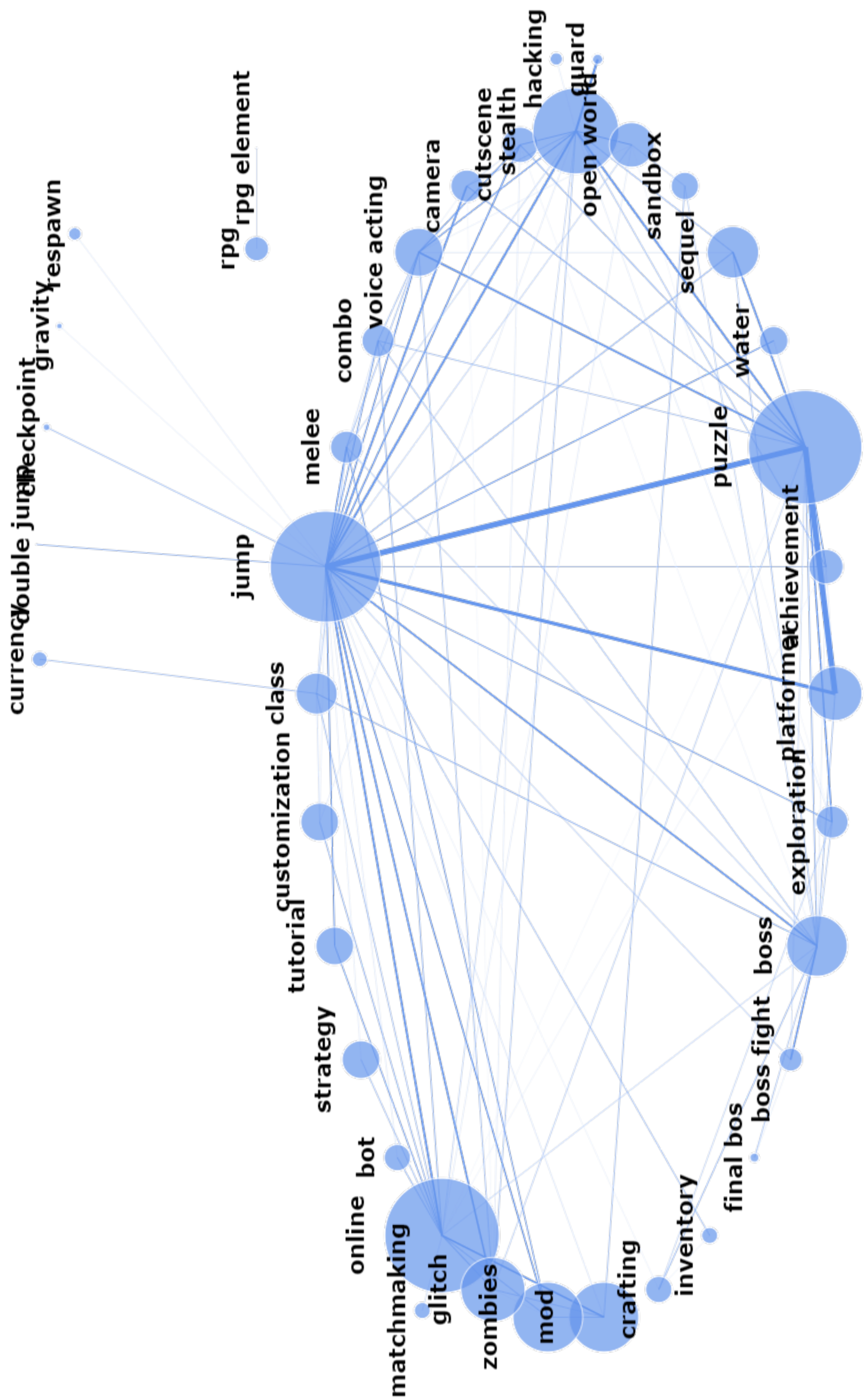
jump
puzzle
platformer
achievement
exploration
boss
boss fight
final bos
inventory
crafting
mod
zombies
glitch
matchmaking
online
bot
strategy
tutorial
customization
class
currency
double jump
checkpoint
gravity
respawn
rpg
rpg element
melee
combo
voice acting
camera
cutscene
stealth
hacking
guard
open world
sandbox
sequel
water

Figure 2.3

2.4.1 Keyword and Keyword-Relations Weights

source	target	weight
platformer	puzzle	6026
jump	puzzle	5841
jump	platformer	4018
stealth	guard	3003
jump	stealth	2754
puzzle	stealth	2682
online	jump	2682
puzzle	voice acting	2589
jump	camera	2581
sequel	puzzle	2528
mod	online	2517
jump	boss	2507
glitch	jump	2504
boss	boss fight	2365
puzzle	exploration	2282
jump	cutscene	2162
jump	melee	2134
zombies	jump	2102
jump	voice acting	2032
voice acting	stealth	1959

Table 2.1: Weighted Edges from the graph

Node Name	weight
online	64945
puzzle	64550
jump	63409
stealth	49330
mod	40552
zombies	40153
glitch	37175
boss	35386
platformer	31282
sequel	30118
voice acting	28453
open world	26668
class	24542
strategy	22754
customization	22441
tutorial	22325
cutscene	21362
achievement	20675
exploration	19733
combo	19695

Table 2.2: Weighted Nodes from the graph

In tables 2.1 and 2.2 are reported the weights of the edges and the weights of the nodes used to build the graph shown in figure 2.3.

Chapter 3

Results

3.1 Validity of the results

The analysis that we performed over the keywords in the space of the Steam reviews of **Action** games highlighted relations between different keyword (and concepts) that can be verified in the real word of video games.

In fact, as we can observe from the word relations graph 2.3, our analysis discovered a relation between the *jump* and **platform** keyword, which is a realistically true relation, since the greatest part of platform games involves jumping.

In the same way we can see a relation between *currency* and *class*, which is a real relation, since quite all the games that involves the concept of game classes do have an in-game currency.

Overall, from table 2.1 we can see the top 20 relations based on their weight, the concepts can have both a positive or a negative meaning, for example *glitch* and *jump* can suggest the developers to pay attention to collisions caused by jumps in order to avoid glitches in the final product.

3.2 Usefulness of the results

Observing those results by the point of view of a startup that is going to develop an Action game but has not decided which characteristics and features it should have yet, this approach seems to fit its needs, since it gives information both on the features of which people talk about the most, both on the most related concept to those features. This can drive the development through a good direction. To sum up, we can analyse our results on the analysis on the reviews of **Action** videogames. From the word cloud we can see that *puzzle* is quite big, this suggests that puzzles in general are important in an action videogame. We can notice that also *online* is large, we can state that, nowadays, the online component is fundamental in every modern game. From thew graph we can see that *puzzle* is connected with *jump* with a strong edge, this may suggest that puzzles should involve also vertical movement.

Overall, this approach can be very useful to a software house since it can simplify the step before the actual development of a videogame, the computation time is low. Moreover, this approach can be further extended in order to help more the development processes.