UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

RECOMENDACIÓN DE VIDEOJUEGOS BASADO EN ANÁLISIS SEMÁNTICO Y MINERÍA DE OPINIÓN

Daniel Yélamos Pérez
Tutor: Alejandro Bellogín Kouki
Ponente: Pablo Castells Azpilicueta

JUNIO 2016

RECOMENDACIÓN DE VIDEOJUEGOS BASADO EN ANÁLISIS SEMÁNTICO Y MINERÍA DE OPINIÓN

AUTOR: Daniel Yélamos Pérez
TUTOR: Alejandro Bellogín Kouki
PONENTE: Pablo Castells Azpilicueta

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2016

Resumen (castellano)

A día de hoy, los videojuegos son una parte considerablemente importante del desarrollo social, económico, así como del sector de entretenimiento de la sociedad moderna. No solo son una gran fuente de ingresos para los desarrolladores, sino también, una enorme fuente de entretenimiento en la vida privada de cada fan.

En esta gran era de la intercomunicación e interacción global, la influencia que tienen los videojuegos sobre nosotros ha crecido a la par de la influencia que tenemos los usuarios en ellos. El gran impacto que las opiniones de millones y millones de usuarios han tenido en distintos productos no solo ha valido para potenciar el alcance comercial de dicho producto. Ha aumentado el estándar de calidad, la definición de géneros, la originalidad y la búsqueda de lo nuevo. Esta gran influencia que ahora mismo los usuarios tenemos sobre los videojuegos y su desarrollo, tiene sin duda alguna un inmenso interés, tanto como para el usuario, que puede guiar más fácilmente al desarrollador, como al experto en marketing que quiere vender más fácilmente un producto.

En este trabajo de fin de grado se explora el uso de opiniones de usuario sobre videojuegos para crear sistemas de recomendación. Los sistemas llevados a cabo no solo son capaces de mostrar información de una manera simple de entender, sino que son capaces de llevar a cabo asociaciones entre distintos productos y separar géneros tanto cercanos como lejanos.

SteamMind (SM) es el nombre del sistema desarrollado en este trabajo de fin de grado. Este proyecto es la suma de tecnologías de extracción e interpretación semántica de datos en un dominio informal, sobre la industria del desarrollo del entretenimiento virtual. Su propósito es el siguiente:

Crear un sistema de recomendación basado únicamente en contenido generado por el usuario de la comunidad de videojuegos de Steam, para ser capaz, una vez proporcionado con una descripción, tan grande o pequeña como el usuario desee, ser capaz de referirle a un juego, o a un grupo de ellos.

Palabras clave (castellano)

Procesamiento de Lenguaje Natural, Videojuegos, Reseñas, Análisis Semántico, Mallet, Word2Vec, Crawler, Minería de Datos, Minería de Opinión, Sistemas de Recomendación, Prototipado Iterativo.

Abstract (English)

Videogames are a very important part of every day's social and economic development, as well as the biggest part of modern society's entertainment sector. Not only it is a considerable source of income for developers, but it is an incredibly big entertainment source of every fan's private life.

In this great global interaction era, the influence that games exert upon us has grown side to side with the influence that we, as users, exert upon them. The great impact caused by millions of user opinions has not only served as a commercial catalyst of the product itself. It has improved a lot of fields within the gaming world, quality standard, genre definition, originality, and the search of something new. This great influence that we now have as users over the whole videogame industry, without any doubt, of great interest, for both users – that can more easily guide a developer or a designer of the game – and the marketing experts – that are constantly looking for ways to improve the sales of their products.

In this bachelor's thesis, the definition of recommendation systems based on user's opinion is explored. The systems that have been developed are not only capable of showing information that is easy to understand, but they are also capable of carrying out associations between similar products and define both wide and narrow genres.

SteamMind (SM) is the name of the system developed in this bachelor's thesis. This product is the aggregation of data extraction techniques and semantic interpretation technologies within a very informal domain, describing the industry which makes the virtual entertainment sector. Its purpose is, then

To create a recommendation system based solely on the input of Steam's gaming community that can be able to refer a user, once given a definition, no matter how short of long, of what s/he is looking for, to a game or a group of games.

Keywords (English)

Natural Language Processing, Videogames, Reviews, Semantic Analysis, Mallet, Word2Vec, Crawler, Data Mining, Opinion Mining, Recommendation Systems, Iterative Prototype.

Special thanks:

To Rafael Perez and Melania Perez. Their advice never unnoticed always appreciated.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

“Video games are ingrained in our culture. Driven by some of the most innovative minds in the tech sector, our industry’s unprecedented leaps in software and hardware engages and inspires our diverse global audience. Our artists and creators continue to push the entertainment envelope, ensuring that our industry will maintain its upward trajectory for years to come.”

Michael D. Gallagher, president and CEO, Entertainment Software Association

1 Introduction

1.1 Motivation

Games are everywhere right now. They are probably one of the biggest industries within the IT domain, and it will most probably continue to be. There are a lot of games, but most importantly, there are a lot of gamers. Lots of different people want different things from games. For some, they want fast movement, action, feel the adrenaline pumping as they make a turn with a car, or shoot a rocket into a group of soldiers. For some, it is the thrill of achieving something really difficult. For some others, it is the quiet time in which they think what is the best move to make. And yet, there is such a very big audience, and so much feedback that can bet tapped into thanks to the World Wide Web and the constant need that this society has of giving its opinion.

Personally, the idea that fueled the project came from the daily routine of a game lover. There is something fascinating about browsing and looking for the game that will entertain you for the next 10, 80 or maybe even 1,000 hours to come, and yet, it is so very hard to find a perfect match for a user. Most hardcore gamers will browse games for a really long time, looking for something that they do not necessarily know it is there. It is a well-known fact in the IT industry that, most of the time, the client does not know exactly what they want, they might have a hint, a slight idea of what they are looking for, but they cannot pinpoint exactly what it is they are looking for. Personally, this project was designed to solve that daily dilemma of "Where is that next game that is going to entertain me?"

However personal this idea may seem, professionally it is quite an interesting topic. The main idea was somewhat vague: using the raw power of massive opinion posts about games that there are currently public, free to get; using that massive information and condense it into something tangible, that will not be based in what a team of 30 people say about their own product, but of what thousands of people commented on; being able to link similar games, not for a genre defined by the makers of the game, but for what people that played the game and paid for the game have to say about it. If any of these goals would be achieved, it would bring a very interesting perspective to the recommendation field.

This project begun as an exploration into the field of "directed recommendation" which could be easily defined as product searching or discovery. The problem relays in finding an accurate description for what a game is, since it is such a multi-layered product. People measure game value in euros per hour invested in the game, in game campaign hours, replayability, and many other terms. Since it is such a hard product to describe, and it can be described in many ways, it would only make sense for the average opinion to be taken into consideration. That is where this project lays.

The goal of this project is then to build an (automatic) definition, of a set of games, that can potentially be used for recommendation systems. This definition does not need to be verbal, it could be numeric, it could be graphic, it could take many forms; the output does

not matter as much as how it is made. The main idea is that the opinion of almost 100,000 users can be condensed into comprehensive models. These models could later be used to point future users to a game that they desire based on cross-referencing their description of their "ideal" game with every model of every game that has been previously analyzed. This project would help understanding the highly complex (and sometimes quasi-random) natural language used by users in these type of systems, which is – still – an open problem in Artificial Intelligence, and Computer Science at large.

1.2 Objectives

The purpose of the project, once again is: **To create a recommendation system based solely on the input of Steam's gaming community that can be able to refer a user, once given a definition, no matter how short of long, of what s/he is looking for, to a game or a group of games.** In order to achieve this main objective, the project is divided in 3 modules, each one addresses a specific part of the problem.

1. **Module I: Crawling and data extraction:** A source of relevant data is to be identified and extracted. This data needs to be abundant and needs to talk just about the product, and not about secondary services such as packaging and delivering. The data is to be downloaded and placed in a file structure or database that would hold the information for future processing, parsing, and querying to/from the proper format that the next part of the project will need in order to perform the analysis.
2. **Module II: The preprocessor:** This module is to parse the data extracted in the first module of the project, and place it in an appropriate format after applying a set of filters that will separate relevant information from non relevant information following an specific set of criteria.
3. **Module III: The semantic analyzer:** The data is to be analyzed to develop a model that can differentiate between different games, or different game genres (groups of games), this will later be used to reference the user back to a game that he/she could be interested in. To make sure that the models are able to properly differ between the games and groups that will be defined. There will be a period of testing in which a small percentage of the data collected in the first stages of the project will be used to test the integrity of the system.

1.3 Report organization.

This report is organized in the following manner. After this short introduction that makes Chapter 1, Chapter 2 will describe the state of the art of the technologies within the project's domain. It will cover the project's domain, the technologies that were considered for the implementation of every part. Following, Chapter 3 will talk about how the project was designed and implemented, justifying the different paths that were taken and the ones that were abandoned. In Chapter 4, the results for the different models and testing methods that were used will be explained and analyzed, justifying the reasoning behind the acceptance or rejection of the models created. Lastly the conclusion and the potential future outcomes of the project will end the report in Chapter 5.

2 State of the art

In this section, the different technological areas pertinent to this work will be discussed, as well as the different technological alternatives that were explored. Furthermore, the reasons behind selecting the techniques for developing this project will be justified.

2.1 The Project's Data Domain

This project was developed to create a unique analysis of the vast amount of information about games from the user's perspective, and not just from a critique's point of view. The main choice for this source of information has been clear from the very start, mainly due to personal experience and interest. Some alternatives, nonetheless, are explored as to provide a source of argument and comparison that would support the choice of Steam as the social network to analyze. The main factors taken into account to select the source of the information to be analyzed are the following:

- Firstly, a vast amount of information was needed to provide a thorough recommendation system, therefore, a source of a huge volume of data that could be easily and publicly accessed was needed.
- Secondly, this information needed a clear context on the game that was under analysis by the user. This context would be given by a broad but simple view of the user's opinion of the game. Text alone would be useless, since this information would have to be sorted out or thrown away if it would not be useful. The data that could give context to this review could be the date, the rating that the user gives the game, the amount of time spent ingame playing, and other user's comments or responses to the reviewer's opinion.
- Thirdly, the platform selected would need to hold a certain reputation and popularity, as to favor the authenticity and relevance of the data.

With all the relevant factors taken into account when picking the most suitable social network laid out and justified, the next section will present each social network apart by these factors, justifying the final choice.

2.1.1 Gamespot, IGN and other online game magazines:

These are some of the most popular game tabloids, both online and printed, through Europe. If we take a closer look at any of their articles dealing with game reviews, a clear pattern is shown.

First of all, most of the user's comments are channeled through reviews created by the chain's employees. This is not necessarily a bad factor, but when we take a look at these responses, it is clear that most of them are not comments about the game itself, but reactions from the audience about a particular article. Secondly, games are classified in a very rigid way, created by a very small amount of employees of the company involved in the review, which limits the reactions of the users. For example, if a user does not agree on a game's assigned genre, or he/she comes up with a very smart term to describe the game, or even a group of games, he/she cannot express this easily enough. Lastly, it can be shown that throughout every site that was under this project's scope (Gamespot, IGN, Meristation) users are asked for their opinion, without a method at hand for the researcher to develop a way of determining whether the information given was relevant or not in a non-semantic way. For example, if there is a review that is one year old about

a product that has just been updated, the review would be of less relevance. This part was very important, since preprocessing and trimming the main corpus to a relevant one was incredibly important for better results.

2.1.2 Metacritic

Metacritic has been the only real alternative as a reliable source of gaming information that could be analyzed, thanks to its clear HTML, and its way to display their information. First of all, it displays both critiques' opinions, and user's opinions. Each user is forced to at least give a comment on their review to justify their opinion. And as it can be seen, the data volume is reasonable (1346 user reviews on a relatively new game[1]). Another advantage of this website is that it is not only for computer games, but for games of every console, and also for movies, shows and even music, which could prove itself to be a fine candidate for testing out this project's output in other areas. As mentioned before, Metacritic has been a great candidate for the data extraction, and it would have been a perfect choice, if not for Steam's immense potential, as we shall show next.

2.1.3 Steam

Steam is a gaming platform that started out in 2003 and proved itself to be the best software distribution paradigm for the gaming industry both then and now. Steam is an immense community of software consumers that is known for its frequent hot deals, offers and for its incredibly active community. Most of the gaming industry for PC is sold, made and distributed through Steam[2], but that is not where its great potential for this project lays. Steam's community is one of the most active online communities, Steam provides a broad selection of features, that allow its users to review games based on a binary system ("recommended" or "not recommended") and to give a review as long as they wish, which, later on, other users can react to in various ways, such as pointing it out as a useful and/or funny review. This may appear simple and easily overlooked for Metacritic's review system based on a 0 to 5 scale, but what Metacritic gains in accuracy, Steam makes it insignificant when looking at their review numbers. Looking at a popular game, within three months from the date of release (users are not allowed to make reviews before they purchase and play a game) it is expected that around 20,000 users will place reviews[3]) and when looking at a popular game that is about one year old, we could gather more than 110,000 reviews[4]. This vast amount of information, as well as how active the users are, make steam simple enough to mine and interpret but incredibly deep and broad information wise. There is also a very important system that Steam implements gracefully, which is the tagging system. Each tag is created by a user, and each user, if he/she wants, is able to vote for a tag. These tags votes, are lately used to index games and search for them using their tags. Once again, these tags are user defined, and user voted, so they are a "review" on its own, but of a very different relevance that will be shown later in this paper.

[1] DOOM, http://www.metacritic.com/game/pc/doom last accessed on 12/06/2016
[2] Steam vs Origin vs Uplay: http://www.gadgetreview.com/steam-vs-origin-vs-uplay-comparison
[3] Dark Souls III, http://store.steampowered.com/app/374320/ last accessed on 12/06/2016
[4] Grand Theft Auto V, http://store.steampowered.com/app/271590/ last accessed 12/06/2016

Figure 2-1: Example of a review in Steam

2.1.4 Comparative study.

In order to show a very simple comparative analysis, the three sources will be consulted for a relatively popular game that went out mid-May: DOOM. Steam, Metacritic and Gamespot magazine (since it is the most popular one at the moment) will be consulted for a broad array of statistics that are related to user generated feedback.

Table 2.1 Comparative Study	Steam	Metacritic	Gamespot
Magnitude of comments	17,240[5]	1397[6]	519[7]
Rating system	Binary	Decimal	none
User text response to user comment	Yes	None	Yes
Rating user comments	Approval, funny	Approval system	Like system
Post date	Yes	Yes	Yes
Edit date	Yes	No	No

When looking at any of these three sources of information it is clear that Steam in plain numbers exceeds the amount of information that can be gathered if we were to add every review found in the two other candidates. Not only that, but its information is relevant to the game, has context within the game, has a context within the community and has a time context, since the dates for both posting and editing a user's review is recorded. This does not only mean that Steam is a source of a lot of information, but that the information itself will be very valuable for analysis and mining purposes.

[5] DOOM's steam page http://store.steampowered.com/app/379720/ last accessed 27/06/2016
[6] DOOM's Metacritic page http://www.metacritic.com/game/pc/doom last accessed 27/06/2016
[7] DOOM's GameSpot page http://www.gamespot.com/reviews/doom-review/1900-6416432/ last accessed 27/06/2016

2.2 Alternative Technologies.

Within the project, a large amount of techniques, software and APIs have been taken into consideration as possible candidates to be used. To be able to clearly explain the extent of each one, the study of the possible technologies relevant to this project has been divided by the different modules of the project itself.

2.2.1 Module I: Crawling and data extraction.

When extracting any kind of data based on the HTTP protocol, there are a few ways that information can be extracted. It needs to be noted that as the websites have been getting more complex in time, there has been a growing complexity of the way the information is displayed. No longer data is displayed plainly on an HTML website, but it now gets filled as the user scrolls down the website (infinite scrolling), and data can be filled thanks to asynchronous processes that make it more difficult to extract the information from the site. Bearing that in mind, a few alternatives for correctly extracting dynamic asynchronous information have been explored. The outcome of this stage should be the HTML of each unit of data to be analyzed in following steps of process.

2.2.1.1 JavaScript

JS (JavaScript) is one of the best candidates for extracting information from the Internet when dealing with AJAX and other asynchronous petitions. The functions can be downloaded and executed locally to be able to make contact with the server and extract the needed data. However, a non-friendly syntax and the fact that no company ever comments nor correctly points out how to call their JS functions and where they are located within the immense maze that the website hierarchy of any complex website is, make it very hard to be able to correctly emulate the behavior of the target website. The advantages of this technology is that once everything clicks, a compact, fast and neat crawler would be at hand. However, the fact that most companies update their services whenever it is needed make this working against the clock, since at any given point in time, the company could simply switch JS functions and all the effort placed into creating that perfect crawler would be spoiled.

2.2.1.2 Selenium

When dealing with dynamic websites and asynchronous calls, Selenium is a rather blunt, yet effective tool that can provide a much needed emulation of an internet user. It is a web browser automaton that will emulate a user navigating a website. This comes in handy, since it would eliminate the problems that rise with infinite scrolling and other dynamic content loading tools in current websites. However, since it emulates a user, the speed that this tool provides is quite low, and since the data that needs to be extracted is gigantic in quantity, this is a poor replacement at best for any Java based crawler that would collect information 10 times faster.

2.2.1.3 HTTP petitions

Since the WWW is mostly based on a REST model, dynamic content can simply be looked as an HTTP petition, with its relevant URL, its parameters, and its response. When looking at any website that loads content dynamically using Google Chrome's debugging add-on,

the next response is observed. Whenever content is loaded, an HTTP petition is launched, and a response comes in shortly afterwards. This can be thought of elementary and easily thrown away as basic information, however, it holds great potential when looking to structure a web crawler. The only thing needed is an agent that could launch HTTP petitions, which are common and varied, especially when looking at Java. Once that agent loads the URL and their parameters, it could iterate over that HTTP petition while altering its parameters in order to receive a constant flow of information. The downside to this way of mining data lays on how it is often put to use, as a way of DDoS attack, since launching multiple parallel HTTP petitions in a very short span of time could overload any server. In order to be able to protect itself from these kind of attacks, most servers implement a black list policy, and once they receive a lot of HTTP petitions within a very short time, they could potentially issue an IP ban that would render the crawling useless. Patience is what mends this flaw, since forcing the crawling agent to wait in between petitions for at least a second could easily prevent this ban, at the expense of more time needed to finish the crawling.

2.2.2 Module II: Data pre-processing.

In order to make sense of the HTMLs downloaded at the previous stage, the relevant data must be extracted, filtered, separated and placed into a suitable format that would allow an easy management of such information. In order to extract the information from such HTMLs, there is a wide array of methods and libraries that could be used. The output of this part of the project should be a txt file, since it is the most common input asked for in the different semantic analyzers that have been studied. These semantic analyzers usually require a very specific pathing in order to be able to process data correctly, word2vec, for example, required a structure such as:

/labeled/game1	/unlabeled/game1
/labeled/game2	/unlabeled/game2

With each of those folders filled with individual pieces of text to be analyzed. Since the examples studied in other technologies also followed a similar format, using a database for storing the information was ruled out, since it seemed like a great investment in terms of working hours for merely formalizing the output of this part of the project.

2.2.2.1 Raw analysis.

Since an HTML is a plain text document such as any other, it can be opened as such and iterated over in order to extract the relevant information. This could be done practically in any language, ranging from Python and Java and getting even into C. This is however suboptimal, since there are plenty of options available for extracting data from HTML that are currently available.

2.2.2.2 JSOUP

JSOUP is a Java based library that allows the user to download HTML code, manipulate attributes, elements and text as well as search for them efficiently and with a very clear syntax. Its syntax is very similar to JQuery (another very popular JavaScript library) which makes any experience with it a great help when coding. It is one of the greatest HTML document scrapping tool available and with efficiency matching its popularity.

2.2.3 Module III: Semantic Analysis.

The semantic analysis technology is the core of this project, since it will help us to make sense of the huge amount of information that it will be provided with. It is important to mention that as it is known, language processing and understanding is a non-solved problem, since not even Google has been able to correctly define a language (make a robot that can speak like a human). Every technology mentioned is but an approximation, a very accurate in case of some, and a not so very accurate in some others, of the solution to the language problem. The output of this final module of the project would be an analysis of the presented data, filtered by different categories and with a clear interpretation. This output could take many forms, that of simple numbers and data providing an analysis of the social gaming network in its lowest complexity level, or a classifier that could divide future reviews into different fields at a higher complexity level.

2.2.3.1 Mallet

Mallet is a "a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text."[8] Mallet is specially designed for document classification and identification of features, after given a sufficient amount of text. It is important to mention that Mallet is specifically based on LSA or Latent Semantic Analysis and the bag of words model, which takes words into consideration, but not its placement in the sentence. To make this clearer, Mallet could take topics out of a pair of sentences, but not its meaning. This is more clearly shown through a simple example. "Real Madrid won last soccer match versus Barcelona" and "Barcelona won last soccer match versus Real Madrid": in these two sentences, mallet would interpret them the same way, it would be able to know that they talk about two soccer teams, but it could not make out who's the winner nor the looser.

2.2.3.2 Word2Vec (Deeplearning4j)

Word2Vec (Word to Vectors) from the Deeplearning4j project[9] is the newest and most popular advance on semantic analysis, fueled by Google. It takes a different approach to the usual semantic analysis, rather than simply relying in techniques as LSA or "Bag of Words", it "vectorizes" words and even sentences and paragraphs through a two-layer neural network and places such vectors into a multi-dimensional map. These vectors are not static, as the neural network is trained, the vectors and the space itself shifts in order to more adequately adapt to what it has learned. The fact that they implement a neural network increments its processing time greatly, since neural networks have a wide array of factors to accurately tune, and most importantly, they have to go through a number of epochs in order to reach a level of learning that is adequate to the topic at hand (not too high, because overlearning is a problem in these systems). As Deeplearning4j states in their website, given enough data to Word2Vec can "make highly accurate guesses about a word's meaning based on past appearances." It displays its information through a cosine similarity, which is the cosine angle between the two pieces of text compared. For example, the list shown in this page will show words associated with "Sweden" based on their distance to this word.

[8] Mallet official website, http://mallet.cs.umass.edu/index.php

[9] Deeplearning4j official website http://deeplearning4j.org/word2vec

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Figure 2-2: Word2Vec output example

2.3 Technologies selected:

When deciding the relevant tool to use in the data mining part of the project, a viability study was developed as to see what would be the most correct way of extracting the data. This viability study is explained further in the design and development section, but the conclusions arisen from that study were clear. JavaScript was a poor choice, since it was quite unstable (3 changes were made to Steam's API in the course of 9 months). Selenium worked quite well in extracting the needed data, however, the mining speed was beyond slow, with the capability of mining only about 5 reviews per minute. Lastly, HTTP petitions were clearly the best choice, since after a bit of tuning and fine adjustment to the latest changes on Steam's website we managed to mine around 80,000 reviews in about 4 days (800 reviews an hour) and with no IP ban ever issued using this method.

In the second part of the project, the data extraction and manipulation, it was clear from the very start that JSOUP's popularity wasn't without reason. Its clear access to the HTML resources were quite efficient in the early trials and its easy syntax made no further research needed.

Finally, the semantic analysis was the most troublesome area, in terms of making a decision. Mallet's reputation as a "topic finder" was a great addition to the project. A set of topics could be used to describe a certain game, and hence the recommendations could be made based on the topics taken out using the same program and a user's description of his/her ideal game. However, Word2Vec seemed a more interesting and flexible option, with not only topic analysis, but the potential to give meaning to vast amount of information without separating topics. It also needs to be noted, that since Word2Vec is newer, more popular, and the roots of the project (shallow neural networks and using words as vectors) were academically more attractive than the more traditional LSA approach to semantic analysis, its use as the semantic analysis seemed a perfect fit for an "out of the box" project such as this. However, during the projects development, it was clear that Word2Vec's maturity as a product was not high enough to adapt to such a complicated topic as this. Word2Vec was originally thought to be the most useful software for the project, but it was later discarded and Mallet was used instead, this will later be explained in Chapter 3.

3 Design and Development

The design behind this project has followed an iterative prototype. Since the problem was made of small parts, it made sense to split the project in stages, each with a well-defined input and output. In each part of the project, functional requirements were added from the start, but the maturity level of the requirement was always taken into account, and so, they could be slightly adjusted depending on how the prototypes varied. Before the system's design and development is thoroughly explained, a brief explanation of the overall project and its functions will be presented:

- SteamMind (SM): This is the name of the whole system, its purpose, once again, is the following:

  To create a recommendation system based solely on the input of Steam's gaming community that can be able to refer a user, once given a definition, no matter how short of long, of what s/he is looking for, to a game or a group of games.

  In order to be able to fulfill its purpose, the project has the following 3 parts/modules:

  - I. The Crawler:
    - This module is in charge of extracting the raw data and putting it into an object, txt file, or database that could potentially be fed to the next submodule of the project. Its purpose is only to collect the data and make it so that it can be treated or filtered by any of its fields.

  - II. The preprocessor:
    - This module will take the data mined by the crawler and will use a number of directives to discard the information that would not be useful. This module will also be in charge of supplying the information to the next layer of the project in an adequate format so that it can be processed without having to modify it in any way that may slow the process down.

  - III. The semantic analyzer:
    - This last part of the project will receive 2 inputs:
      - Input I: The prepared, preformatted data from the second layer of the project. It will use it to create a model that can map different combinations of words to a game, or a group of games.
      - Input II: A user's description of the product that s/he is looking for. It will then use that information to refer the user to one of the games that it analyzed before.

3.1 Module I: The crawler.

The crawler's function is to collect data and place it into an object. In order to explain more thoroughly each of its functions, the definition of this part of the product will be divided in different requirements, functional and non-functional. The functional requirements describe the behavior of the system and how every input is processed and turned into an output. The requirements will be displayed as follows:

- **Name/ID:** This field allows a correct identification of each requirement, so that in future references it will be easier to classify and consult them.
- **Description:** this field will explain what this requirement is to fulfill and in what way.
- **Pre-conditions:** this field describes the conditions to be met within the application so that this requirement can be evaluated.
- **Post-conditions:** this field will define the output of the function described by this requisite.

3.1.1 Requirements:

3.1.1.1 Functional Requirements:

The functional requirements of the crawler are as follows:

ID	**FRA01**: Steam integration
Description	The system must be able to communicate with the site www.steampowered.com so as to be able to extract the review data.
Pre-conditions	The system in which the application is run must have an internet connection.
Post-conditions	The application does not raise any exceptions due to a failure in connection.

ID	**FRA02**: Game list collection
Description	The system must be able to crawl the game category "most popular steam games" and extract the product IDs of each of the most popular 18 games.
Pre-conditions	Requirement **FRA01** has been met.
Post-conditions	The application collects a list of URLs or game ids and stores them in memory to be able to be forwarded to the next part of the program

ID	**FRA03:** URL builder
Description	The system is able to come up with the list of URLs of every game on the list produced by **FRA02.**
Pre-conditions	The requirement **FRA02** has been met.
Post-conditions	The application will be able to access correctly every steam game URL based on the **FRA02** list.

ID	**FRA04:** Game review crawler.
Description	The system will be able to crawl the URL of the game provided by **FRA03** and extract the URL from the X (this number is flexible) first most popular user reviews.
Pre-conditions	Requirement **FRA03** has been met.
Post-conditions	The application successfully collects a list of review URLs and dumps them into a text file for later processing.

ID	**FRA05:** Game review data extractor.
Description	The system will be able to extract the relevant fields of data from each of the review URLs.
Pre-conditions	Requirement **FRA04** has been met.
Post-conditions	The system is able to collect all the data into a List<Review>.

ID	**FRA06**: Lost URL detection
Description	Users are able to delete their reviews, because of this the review can be deleted after it has been fetched by the game review crawler (**FRA04**). Hence an error system that checks for a null or lost review is needed
Pre-conditions	Requirement **FRA04** has been met.
Post-conditions	The system is able to point out which reviews have been deleted and are no longer available.

ID	**FRA07:** txt dump file
Description	The system is able to dump all the relevant data into a txt file so as to be ready for the pre-processor.
Pre-conditions	Requirements **FRA06** and **FRA05** have been met.
Post-conditions	A text file with all the information is correctly generated.

3.1.1.2 Non-Functional Requirements:

The non-functional requirements are those which describe the properties and qualities that the system must have. These are not part of the functionality, but they dictate the ways that each of the functions must be carried out so as to obtain a desired level of quality.

ID	**NFRA01:** Stability
Description	The system must be stable enough so as to maintain a proper execution during long periods of time in which the crawling and web-scrapping must be carried out.
Priority	High

ID	**NFRA02:** Mining speed
Description	The system must be able to mine the required data in a fast way, relative of course to the huge amount of information that is to be collected and the bottlenecks of the different software systems in which it may be based on.
Priority	High

ID	**NFRA03:** Portability
Description	The system must be able to be run correctly under every SO, Linux, Mac, and Windows so as to provide the highest possible compatibility to mine from different systems.
Priority	Medium

3.1.2 General design:

This module has been developed in a modular setting that allows a very clear description of the whole process that the crawler needs to carry out. The submodules are as follows:

- **Crawler Module I: Game list**
  - This module allows to fetch either the URL or the ID of each steam game that is to be crawled. In order to create a stable list of games from which to extract the reviews, the steam category "most sold games" was chosen. This was a design choice, since the most sold games would have a higher number of reviews.
  - **Input:** the URL where the game list will be fetch from (see later).
  - **Output**: a list of URLs or IDs that will be used to crawl the game reviews.
- **Crawler Module II: Game review fetcher:**
  - This module lets fetch the different reviews up to a maximum defined by a local variable (10,000 reviews) by their reference or URL.
  - **Input:** The list fetched from module I.
  - **Output:** A list of URLs to be scrapped by the next module in either a .txt file or an object (List<String> would suffice).
- **Crawler Module III: Review page-scrapper.**
  - This module allows to:
    - Read from every field the relevant data.
    - Store data in an object.
    - Have a toString method so as to dump that information in a file for later use.

To be able to justify this design, a concise analysis of the Steam website will be provided. First of all, these are all of the relevant URLs:

- Type I: Game list URL (from where the game list will be fetched): http://store.steampowered.com/search/?filter=globaltopsellers&os=win#sort_by=Reviews_DESC&category1=998&os=win&filter=globaltopsellers&page=1
  - This is a rather complex URL, but it can be understood easily. It is the list of most sold games ordered by reviews and showing only games (not taking into account downloadable content (DLCs and expansions).

- Type II: Game URL (Example: Portal 2):
  http://store.steampowered.com/app/620/
    - As it can be seen, the game URL is easily fetched since it happens to be indexed by an id at the end of the URL. After a bit of research it was found that this ID is unique to each game and so fetching this id would be the first step to create the crawler.
- Type III: Game review display URL:
  http://steamcommunity.com/app/620/reviews/?browsefilter=toprated
    - As shown, the URL maintains the appid structure from the game URL, and it simply adds the /reviews/ part to the game URL mentioned before. Also, it can be seen that the /?browserfilter=toprated indicates that it will show the top rated reviews first. It would seem like this would be a perfect starting point for the crawling tool.
- Type IV: Game review URL:
  http://steamcommunity.com/id/weffal/recommended/620/
    - As shown, the reviews are assigned to the user (in this case *wefall*) and they do not carry an individual ID, which means that our system cannot construct the URLs from the game ID alone. It must also either know every user that has reviewed the game (which is not easily accessed information) or fetch the review URLs from the game review display URL.

The easiest way to fetch the URLs from the reviews seemed to be fetching from the game reviews display URL, which was publically shown and accessed easily from constructing the URL (no crawling needed). To show how this was done, the different pages from which the information is going to be fetched will be displayed in HTML code. Let it be noted that the HTML page is around 4,000 lines long, so only the relevant data will be shown as to provide an insight into what information the crawler would fetch and from what fields.

First of all, the game IDs could be fetched from a structure from the Type I URL such as the one that follows:

```
                <div style="clear: left;"></div>
            </a>
                    <a href="http://store.steampowered.com/app/395180/?snr=1_7_7_globaltopsellers_150_1"  data-ds-appid="395180" onmouseover="GameHover( this,
event, 'global_hover', {&quot;type&quot;:&quot;app&quot;,&quot;id&quot;:395180,&quot;public&quot;:1,&quot;v6&quot;:1} );" onmouseout="HideGameHover( this, event,
'global_hover' )" class="search_result_row ds_collapse_flag" >
                <div class="col search_capsule"><img src="http://cdn.akamai.steamstatic.com/steam/apps/395180/capsule_sm_120.jpg?t=1465470573"></div>
                <div class="responsive_search_name_combined">
                    <div class="col search_name ellipsis">
                        <span class="title">Arma 3 Apex</span>
                        <p>
                            <span class="platform_img win"></span>                        </p>
                    </div>
                    <div class="col search_released responsive_secondrow">Jun 2016</div>
                    <div class="col search_reviewscore responsive_secondrow">
                                    </div>


                    <div class="col search_price_discount_combined responsive_secondrow">
                        <div class="col search_discount responsive_secondrow">
                            <span>-20%</span>
                        </div>
                        <div class="col search_price discounted responsive_secondrow">
                            <span style="color: #888888;"><strike>29,99€</strike></span><br>23,99€                        </div>
                    </div>
                </div>
```

Figure 3-1: Steam Homepage Source Code

As it can be seen, the id could be fetched straight from the <a href…data-ds-appid field. Also, the name could also be fetched from the <div class=”col search name ellipsis”> and the class=”title”.

The type II URL (game URL) did not have a use in the crawling program, since it only held game information, so the type III URL was explored only to find a very common use nowadays of dynamic content: the infinite scrolling. This is a common solution for web developers to avoid loading too much content in one single URL, so what they design instead is a JavaScript script that could detect when a user is at the bottom of a page, then an AJAX function is executed to dynamically load content into the page, but only so as the user kept scrolling down.

The infinite scrolling was a very hard problem to tackle. It did not allow JSOUP to fetch the whole reviews with only loading the type III URL, which was a huge problem, because the information was able to be accessed publically, but the human interaction was needed. Here is where the iterative prototype started, since the requisites were into place and the only thing avoiding for the process to continue was this stone in the path. Instead of doing a viability study for each solution which would take considerable time and could potentially be unable to yield the necessary resources, different routes were explored within the iterative prototype paradigm.

3.1.3 Prototype I: JavaScript crawler.

The first exploration was the most obvious one, since the request to load the page was dynamic, and it was called by a small JavaScript function on the website, it was thought at the time that JavaScript could be run locally as to imitate the behavior of the website and receive the data in a JSON file or something similar. Since everything that is on the site must be publically shown, especially if it is locally executed (such as JavaScript) the function was available to the development team. However, no documentation was provided for this function. The arguments were shown without clear indications on what they were doing or what their purpose was. During the first semester of the project, this option was thoroughly explored. Different versions were developed as to execute JavaScript in different ways, however, at the moment in which it was almost ready (URLs were successfully fetched and placed in a txt file) Steam decided to update their website. That brought a change that rend the work done until that point useless, since they almost deleted half of the JS functions that they called. As a result of this update, this side of the project was abandoned and left for a somewhat unorthodox solution: Selenium.

3.1.4 Prototype II: Selenium crawler.

Selenium is a software testing framework that was specially designed for testing web applications. It allows web browsing automation under a clunky user interface and a very slow pace. Setting this crawler up took less than a week. The program instantly opened 5 to 10 different browser windows (each belonging to a single game) and it was able to scroll down while collecting each of the user’s name and review URL. This allowed for collecting both the username of the reviewer (used to create the URL for the review) together with the game id, and the review URL itself. This method, however unorthodox it may have been, worked successfully and it allowed a perfect fetching and scrapping of the needed data. However, even when using 10 browsers at the same time, the download

speed was very low, achieving only around 200 reviews per hour. This was of course under no consideration, since mining every piece of relevant data would take weeks. Also, for display purposes, this clunky method seemed sub-optimal. To be able to take this project to the next level, an intensive study of the infinite scroll was carried out. In the end a small forum post in a web development website suggested that every piece of the internet is based on the REST model. Which of course meant that every piece of data that is exchanged is based on an HTTP petition with an URL and a series of arguments that would tell the server how to respond. That was the key for solving the infinite scrolling problem.

3.1.5 Final Prototype: Development of HTTP-GET based Java crawler.

During the intensive study of the infinite scroll it was pointed out that everything that happens in a website has a HTTP petition attached to it. Using Chrome's debugging console, as the game review page was scrolled down, it showed an outgoing HTTP petition that held the solution to the infinite scroll problem. If this petition could be sent while altering its parameters, it could potentially provide the URLs for each review within a game. With this HTTP GET petition at hand, a small Java prototype was tested to check the speed of mining a single game, Darksouls III. The Java prototype successfully mined 10,000 reviews in 2 hours before getting an IP ban. This was the stepping stone in which the whole crawler module was built.

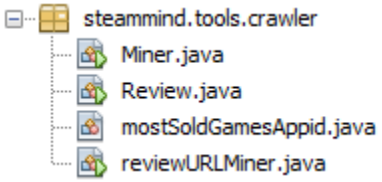

Figure 3-2: SM crawler structure

The formal design of this class is presented next. The classes within this package maintain a very simple, yet powerful structure that could potentially allow for a greater expansion into other parts of the Steam website. The package structure will be explained from the innermost layer, to the uppermost. Since that will allow a greater understanding on what every layer takes from the one before and what it takes from the one after.

In order to introduce the whole package, a class diagram is provided.

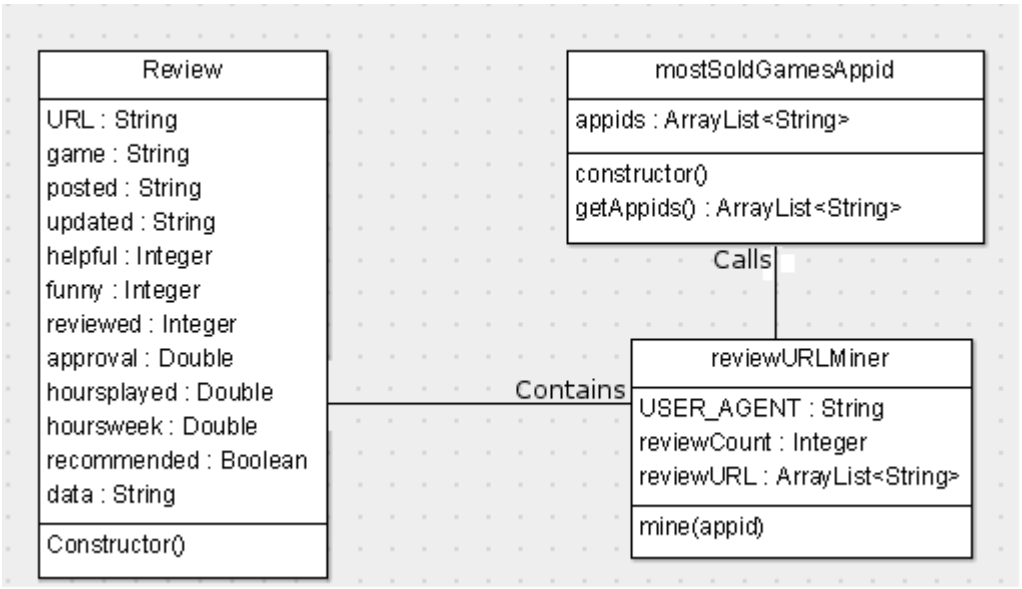

Figure 3-3: Package structure

3.1.5.1 The Review Class.

First of all, a picture of a review will be displayed, so as to show where the information was taken from:

Figure 3-4: Steam review example.

The class review, is the undermost piece of the clockwork that makes the crawler. Its function is simple, to hold the information of a review, initialized by its link. In order to do this, the constructor receives a link, and uses the methods from the JSOUP package to trim down the information and save it in a data structure containing information relevant to the whole process.

url: The original URL in a string format from where the review was fetched.
game: A string containing the name of the game, needed for the classifier.
posted: A string containing the date in which the review was posted.
updated: A string containing the date in which the review was last edited.
helpful: the number of users that rated the review helpful, as an integer.
reviewed: the number of people that reviewed the review.
funny: the number of people that thought the review was funny as an integer. This field was kept in order to be able to filter by it on a later time.
approval: it is a double calculated by dividing **helpful** by **reviewed**.
hoursplayed: a double storing the amount of hours that the reviewer played the game, taken from the website review.
hoursweek: a double storing the amount of hours that the reviewer played the game within this week.
recommended: a boolean storing whether the user recommended the game or not.
data: a string containing the text of the review.

From all the information that the review collected, only a portion of it was dumped to the data file, the rest were thought of minor relevance and left for future stages of the project. This is an example of one of the reviews dumped from the review:

ID:12
Total_War_WARHAMMER
First Campaign in this game, hard difficulty. I pick Chaos, with Archaon as a legendary lord. Turn 2, I attack a town, I make my army suicide itseft, rushing straight into a trap. I lose the campaign, but... Archaon died, so The End Times will never happen. 10 out of 10, would prevent the Old world to enter the Age Of Sigmar again.
approval:76.86310904872389
funny:310
helpful:328
posted:Posted: 28 May @ 8:48am
recommended:true
hours played:21.9

Based in this dump, the most important fields are of course the recommendation, the approval rate and the text itself, together with the number of hours that the user has played. The logistics of the analysis will be explained further into this report.

3.1.5.2 The reviewURLMiner class:

The purpose of this class is to send the HTTP-GET petition mentioned in the introduction of this section in order to fetch a determined number of review URLs and place them within its structure. In order to do this, the HTTP get petition that refreshed the infinite scroll from the review page was studied and was put through a few iterations to test how its parameters were altered. The petition was as follows:

```
String url = "http://steamcommunity.com/app/" + appid + "/homecontent/?"
        + "userreviewsoffset=" + count
        + "&p=" + counter
        + "&itemspage=2" + counter
        + "&screenshotspage=" + counter
        + "&videospage=" + counter
        + "&artpage=" + counter
        + "&allguidepage=" + counter
        + "&webguidepage=" + counter
        + "&integratedguidepage=" + counter
        + "&discussionspage=" + counter
        + "&l=english"+ "&appHubSubSection=10" + "&browsefilter=toprated"
        + "&filterLanguage=default&searchText" + "=&forceanon=1";
```

Highlighted in pink, there is the app id from the game that the agent would be mining reviews from. Highlighted in yellow, there are the parts of the HTTP petition that did not change after debugging the code 5 times in a row within the same page (activating the infinite scrolling 5 times in a row, collecting and comparing the outgoing HTTP petitions). Highlighted in blue, we show the ones that had an argument that always changed at a constant rate. Finally, highlighted in green we observe the part of the URL that dictated the offset of the review table that was consulted. Knowing how the petition worked, and using the package javax.net.ssl.HttpsURLConnection, the petitions could be sent.

In order to maintain the data, the next information was kept:

USER_AGENT: A string that would hold the agent that we would impersonate in order to mine the reviews, in this case Mozilla/5.0
reviewURLs: A list of strings that would hold the URL that the HTTP petition would return. This would later be passed onto the class Review one by one in order to extract the information needed.
reviewCount: this constant was used to determine the number of reviews to be mined, after this number was reached, the agent could stop mining reviews from the game.

In order to mine the relevant data, this class only needed the game id (highlighted in purple above) to start mining information, since every other parameter depended on the number of times the HTTP petition was sent beforehand. So the class simply needed of a method **mine** that would receive an appid, and would mine **reviewCount** review URLs using the HTTP petition that would be built iterating over the same appid with different parameters. The response was simple enough that a simple pattern matcher was used to extract the URLs, no JSOUP class was needed to parse anything else.

3.1.5.3 mostSoldGamesAppid class:

The almost outmost layer of the app makes a connection to the Type I URL in order to extract the most sold games. The class is barely 60 lines long, it simply makes a JSOUP connection to the Type I URL, and downloads the HTML present in such page. From there, it was observed that the "data-ds-appid=\" field of one of the HTML fields contained the appid for each game that was within that HTML division. Extracting such appids was as simple as compiling a pattern such as the one described before, and all the appids were listed in clear ints.

3.1.5.4 Miner Class

This class is the crawler's main class. Its purpose is to call every other class in order to fully scrap about 100,000 reviews. It has a few parameters that can be modified before the execution so as to modify its behavior. Such as the parameter "mined" which determines whether the review URLs have already been collected or not, and if they have been beforehand, it skips this step, making it easy to do a two-step mining.
The main is structured as so:

1. Check if the list of reviewURLs has already been collected.
    a. If not:
        i. Declare a mostSoldAppIds class, and get the appids for every game in that type III URL.
        ii. Declare a URLminer, and call its mine() function once per appid.
        iii. Notify by standard output the number of reviews URLs that it holds.
        iv. When it is done, write every URL into a "reviewURLs.txt" so as to be able to read from it for the next step of the data mining.
    b. If it has been collected:
        i. Read the URLs from the file and store them in an ArrayList.
2. For every review URL in the arrayList collected above:
    a. Create a Review Class with the URL
        i. Let it be noted that the review collects data by itself, using only the review URL.
    b. Dump the review.soString() output to a file.

3.2 Module II: The preprocessor.

The preprocessor that was originally thought of was a lot more complex that the simple class that ended up making this part of the project. Originally, it was designed to create partitions for both testing and training, combining different parts of the data depending on length, gameplay time, and to be able to filter by every attribute collected. However, after switching back and forth from different semantic analyzers, it became clear that this part of the project was too ambitious for this scale. The fact that it does not fulfill its original purpose does not make it less important. This is the parser and selector of data, it places information in a way that every technology used could interpret and analyze. The general design of this part did not fully follow an iterative prototype on its own, this part was adjusted as the semantic analyzer needed fit. As the semantic analyzer went through its own iteration the preprocessor needed to be adjusted and modified to alter at the same pace as the semantic analyzer.

3.2.1 Requirements:

3.2.1.1 Functional Requirements:

The functional requirements of the second module are as follows:

ID	**FRB01**: Data low level parsing
Description	The system must be able to read plain text in UTF-8 format and remove any characters that would not bring any useful information to the semantic analyzer.
Pre-conditions	The Module I has been executed correctly and the reviews are in a group of files following the review dump file syntax.
Post-conditions	The application is able to store the information without any special characters, such a hearts, accented words, Japanese kanji, ascii art and other non UTF-8 characters.

ID	**FRB02**: Review integrity
Description	The system must be able to know when a review is useful or not. This will be done through a list of chained requirements that may or may not vary depending on the restrictions that are used later on the semantic analyzer
Pre-conditions	Requirement **FRB01** has been met.
Post-conditions	The application successfully applies a chained condition in order to determine whether a review should be added to the partition or not.

ID	**FRB03**: Review format
Description	The system must be able to collect the data and parse it into different formats for any semantic analyzer that could potentially analyze data.
Pre-conditions	Requirement **FRB02** has been met.
Post-conditions	The application successfully puts files in any file structure, hierarchy, or format readable for any semantic analyzer that the project needs.

3.2.1.2 Non-Functional Requirements:

ID	**NFRB01:** Speed
Description	The system must process and assemble the different formats and file structures properly and at a reasonable speed.
Priority	High

ID	**NFRB02:** Capacity
Description	The system must be able to hold all of the information without overburden the computer in which it is been run on.
Priority	High

ID	**NFRB03:** Scalability
Description	The system must be able to withhold the future improvements or updates that it could need at a future date.
Priority	Low

3.2.2 General review quality specs:

The formatter is built to fit a general purpose, which is to trim down the amount of data that has been collected, so as to take only the relevant data, and the second one, which is to place it in an adequate format. This part is dedicated to explain and justify the design of the first part, the one that creates different partitions of information.

First of all, when looking at reviews, there is one thing that is very clear. Steam users are not formal critiques. Here, there is an example that perfectly captures the steam community's spirit. http://steamcommunity.com/id/ShariganXII/recommended/374320/ (shown next page).

However, amusing these reviews might seem, they do not bring any kind of semantic meaning to the review. In terms of analysis they are an anomaly, and one that would insert a lot of noise into the system. However, they make barely a 2% (actual fact taken from the pre-analysis of the data) of the data taken from the website. In order to avoid these kind of reviews, ASCII art, Japanese or Chinese characters and many other characters that could potentially pose a threat to the stability of the system, the preprocessing rule number one was created:

Figure 3-5: Steam informal reviews.

Preprocessing rule I: every character that was not a letter, a hyphen, or an underscore, is to be taken away from the sentence.

This includes accented letters, numbers, since they pose no semantic value, and brackets and such. However, that was not the only problem that the informal status of Steam's community posed. Reviews that held very little content, were also a problem, as it can be shown below:

The user Lupo wrote this review talking about The Witcher III
"I hope one day I get amnesia so I can play again to full effect."[10]

The user Clementine wrote this review, talking about the Witcher III as well:
"537 hours and still continue playing. That's your answer for how good it is."[11]

The user Lewiz also talked about the Witcher III in a similar manner:
"The more you play other games, the more you love Witcher 3"[12]

The last example, taken from a Dark Souls III review, will also speak for itself:
"Hmm... Mmm..."[13]

These types of reviews are also considered to be of very little semantic value, and hence the second preprocessing rule was stablished.
Preprocessing rule II: every review that is no longer than a given length is to be removed from the partition.

The last review that was shown, also indicates a very small problem that some of the reviews have, which is word variation. That is why, the third preprocessing rule stated that:

Preprocessing rule III: if there are no more than 5 different words in a review it is to be discarded.

Which is also with clear justification, first of all, a review such as the one shown last, has no use whatsoever. The ones before it, state very little, but if pieced together may be able to account for some semantic value, however, a very small repetition of words will not hold any significance and hence, must be discarded. Lastly, a very long chain of characters with no spaces whatsoever in between, does not hold any meaning either, and hence the last preprocessing rule states:

Preprocessing rule IV: If there are no spaces in a review, it is to be discarded.

[10] Lupo's review on "The Witcher III": http://steamcommunity.com/profiles/76561198041889474/recommended/292030/
[11] Clementine's review on "The Witcher III": http://steamcommunity.com/profiles/76561198008686104/recommended/292030/
[12] Lewiz's review on "The Witcher III": http://steamcommunity.com/id/124123534645361234124312/recommended/292030/
[13] Reinski's review on Dark Souls III: http://steamcommunity.com/id/Reinski/recommended/374320/

3.2.3 Word2vec format:

At first, word2vec was thought of to be the most adequate analyzer for the project, and so an adequate formatter was built. The word2vec format was very simple. Each topic, game, or subject to be identified must have a separate folder, and the corpus would be split into two different sections, labeled and unlabeled. The labeled partition contained the game folders and each game folder contained the reviews that were to be used specifically for the training of the model. The unlabeled partition in the other hand, held exactly the same folders with exactly the same names, but the reviews in them were used for the testing partition. In order to run different tests, two partition generators were created, one that read the whole review data and separated them by games, and another one that separated them into positive recommendations and negative recommendations. The file structure was as follows:

- Classifier by games:
  - Labeled
    - The Witcher 3
      - Review 1
      - Review 2
      - Review 3
      - ….
      - Review N
    - Dark Souls 3
    - …
    - Rocket Legue
  - Unlabeled
    - The Witcher 3
      - Review N+1
      - Review N+2
      - …
      - Review N+M
    - Dark Souls 3
- Classifier by recommendations
  - Labeled
    - Recommended
      - Review 1
      - Review 2
      - …
      - Review N
    - Not Recommended
  - Unlabeled
    - Recommended
      - Review N+1
      - Review N+2
      - …
      - Review N+M
    - Not recommended

Where each black square is a folder with exactly N reviews in the labeled area and M-N reviews in the unlabeled area. Partitions were created in different percentages, with the lowest training percentage of 70% to a testing percentage of 30% and a highest training percentage of 90% and a testing percentage of 10%.

3.2.4 Mallet format

For causes yet to be explained (see Section 3.3, a change in the semantic analyzer was needed. In order to create a format suitable for this new type of software, a few different examples from Mallet's official website were examined. Since Mallet's purpose was very different from Word2Vec, the format was a lot simpler. It only needed raw text in order to function properly, in no given format, so the reviews were simply split into different games for clarity purposes, and they were left in a single folder. A much cleaner format that allowed for very quick on the go tweaks, like removing games that could introduce noise in the system. The file structure for this format creator was as follows:

- Mallet folder.
    - Partition 1
        - GameReviewsI.txt
        - GameReviews2.txt
        - ….
        - GameReviewsN.txt
    - Partition 2
        - Game Reviews1.txt
        - …
    - Partition 3
        - Game Reviews.txt
        - …

3.3 Module III: The Semantic Analyzer.

The semantic analyzer is the most important part of this project, since it condenses all the information taken by the first module, and places it into a structure that can be comprehended. Afterwards, this very same structure will be used to evaluate the new text input, which could be a user's description of the game that he would like to find. This second evaluation will allow the model to make a recommendation based on what the other 80,000 users had to say about the particular game that they wrote about. In order to come up with the most suitable technology for this kind of analysis, two main technologies were used. At first, it was thought that word2vec would be the most suitable technology, and that it would bring prestige to the project, since it is a new technology that is still in development, with a very active community and a lot of developers behind. Mallet however, has been around for a few years and it has a more classic approach to the subject of semantic analysis using LSA and topic modeling instead of the complex geometrical evaluations that word2vec brings to the mix. To make it clear, during a period of almost 2 months, word2vec was used to try to develop an effective model that would suit this corpus. However, after many different issues that were raised during the development of the project, it was clear that it may not be totally suitable for the domain of the project. These issues will be further explained in a later time within this chapter.

After failing at successfully implementing a model with word2vec, Mallet was tried out and the results were processed at a much faster rate, and were a lot more precise. Without further explaining, the requirements of the system that was to be created will be listed and justified.

3.3.1 Requirements:

3.3.1.1 Functional Requirements:

ID	**FRC01**: Stop words
Description	The system must be able to take in a list of stop-words, which are common words that are frequently used and do not bring any significance to the text to be analyzed. These words can and should be dynamic in order to tune the model.
Pre-conditions	The Module II has been executed correctly and the reviews are in a group of files following the correct software syntax (either Word2vec, or Mallet)
Post-conditions	The system is able to ignore every word listed in the stop words list.

ID	**FRC02**: Unattended learning
Description	The system must be able to come up with a model suiting input corpus. This model's specifications could vary depending on the current approach of the project
Pre-conditions	Requirement **FRC01** has been met.
Post-conditions	The system creates such model and places it in a text output that can be interpreted.

ID	**FRC03**: Evaluation
Description	The system must be able to create a list of results that can be evaluated.
Pre-conditions	Requirement FRC02 has been met.
Post-conditions	The system is able to either interactively or automatically be tested and compiles a list of results.

3.3.1.2 Non-Functional Requirements.

ID	**NFRC01:** Speed
Description	The system must be able to come up with a model in a reasonable amount of time, no less than 3 hours of processing would be a reasonable.
Priority	High

ID	**NFRC02:** Stability
Description	The system must not, under any circumstance, cause an exception that could stop the processing, since it would damage greatly the speed of the project.
Priority	Very High

3.3.2 Prototype I: Word2vec-based SA.

The first semantic analyzer that was tried out, was Word2Vec's paravec technology. Word2Vec is "a two-layer neural net that processes text. Its input is a text corpus and its output is a set of vectors: feature vectors for words in that corpus."[14] What the developers from the Deeplearning4j team state is that word2vec turns text into a numerical vector form that deep neural nets can process and analyze. The usual output of a word2vec program is a dictionary or map in which each word is mapped specifically to a vector, and later, those vectors can be queried to find their relationship with other words in the same form. That relationship is measured by the cosine similarity between two words, which means that a word A mapped as a vector has no relationship with another word B also mapped as a vector so as long their intersecting angle is one of 90 degrees. As the angle gets smaller, the relationship is stronger.

Knowing the basics of word2vec, the experiments on the prototype that was created in order to analyze text using this technology will be explained. First of all, word2vec provides a broad repertory of examples, that can be used to build software based on

[14] Word2Vec's official website http://deeplearning4j.org/word2vec

them. The one that was more closely examined was the "ParagraphVectors ClassifierExample.java"[15]

This small Java program was said to be able to classify documents, when given them in the proper format (see Section 3.2.3 for more information about word2vec format) it would be able to learn different patterns from different topics in text. So if fed with enough documents of finance and health, it would be able to learn to distinguish them. Later, when given a document with no label, it would process it and be able to find the relationship to the other labels. The relationship would be mapped in cosine distance and the output would look like this:

```
Document 'unlabeled_health' falls into the following categories:
    health: 0.29721372296220205
    science: 0.011684473733853906
    finance: -0.14755302887323793
```

This originally was thought able to distinguish between game genres, "recommended" reviews or "non-recommended" reviews, games, or even positively classified reviews (as in reviews that are highly valued due to the "useful" parameter of such review). In order to test this classifier, a first implementation of the example was made. This example had a very small variation, and it was supplied with a huge amount of examples: the original only supplied with 3 categories, with 10 documents for the train partition and 1 for the test. For this test, it was fed with 7 games, 1,000 reviews per game in the training partition, and 300 reviews for each game in the test partition. However, the program crashed before even starting processing from a very odd exception: "Exception in thread "main" java.lang.IllegalArgumentException: Length must be >= 1" This stopped the project for a few days, and contacting word2vec's development team was needed. Thankfully, they were quite active and during a period of about two days, a cause for the exception was searched for. The filed issue can be found here in case of future references: https://github.com/deeplearning4j/deeplearning4j/issues/1623

The user Raver119 assigned to paravec development stated:

"So here's the problem:
You should train your model on both labeled/unlabeled data, so all words available in vocab. And your exception appears because not a single word from document was found in vocab."

This was done before preprocessing rule IV was assigned to Module II, hence, a review was found that had the next text:
"*CLAP**CLAP**CLAP**CLAP**CLAP**CLAP**CLAP**CLAP**CLAP**CLAP*"
Which was the cause of the exception.
This was only one of the few problems that the project had when processing the data. First of all, the processing speed was very slow, one full run over what will be defined as partition A took over 8 hours to fully process everything:

[15] ParagraphVectorsClassifierExample: deeplearning4j's github https://github.com/deeplearning4j/dl4j-0.4-examples/blob/master/src/main/java/org/deeplearning4j/examples/nlp/paragraphvectors/ParagraphVectorsClassifierExample.java

- Partition A:
  - 20 epochs
  - 7 games,
  - 1,000 reviews per train partition 7*1,000=7,000 reviews train partition
  - 300 per test partition. 7*300 =2,100 reviews test partition

This would not be a cause for concern if only the results were as the developers explained. However, there was a second problem behind all that processing time. The second problem arose when the first testing was made. There was one group that was more dominant than the others. In the partition A format, one of the games would be assigned over 95% of the reviews from the 2,100 review partition. Or so it would look when processing the output of the simple example program that was given. In order to be able to make a decent count without having to manually count every output, the class Results was implemented only for the word2vec format.

3.3.2.1 Result class:

The result class was implemented to be able to get the usual word2vec output (shown in 3.3.2) and transform it into a table, or organized structure. To understand the output of word2vec better, there are a few very important facts:

1. For every test review the program knows what it belongs to. This will be the label of the Result.
2. It tests the average cosine distance to every class that has been recorded in the training partition. So for each test, there will be X cosine distances calculated where X will be the recorded labels at the Train partition (In case of train partition A, there will be 7 cosine distances calculated)

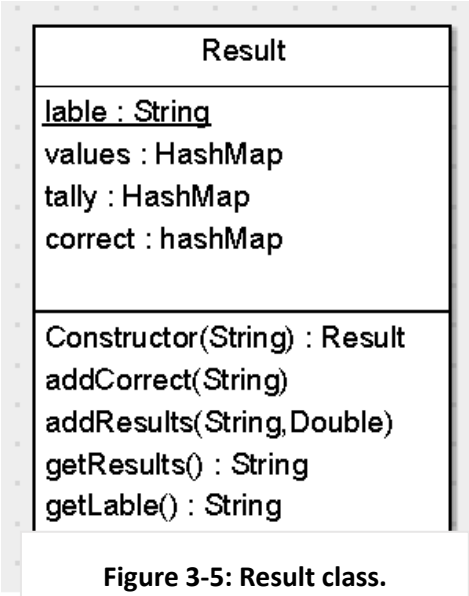

Figure 3-5: Result class.

One result class was created in execution time per game, and was assigned its name for the **label** field. The **values** field was a **HashMap<String, Double>,** in which the Double was the sum of the cosine distance of every test review that was processed by the class, and the Label was the Group that the cosine distance was calculated to. The **tally** field was a **HashMap<String, Integer>**, and for every value added up in **values** it would add one more, it was later used to create an average cosine distance per group. For example, the next hashmaps (column 1 and 2) that come from a set of test reviews that has 4 reviews would be used to create the results:

Table 3.1 Result class example	Values	Tally	Result
Witcher 3	4	4	1
GTA V	3	4	0,75
METAL GEAR R	1	4	0,25

Lastly, the field **correct** was another **HashMap<String, Integer**> that would increment the value of each tag if the review yielded the closest cosine distance (positive one would be the closest, and negative one would be the furthest). The logic behind this is that if a review from a label “Witcher 3” comes and its closest cosine distance is the label “GTA V”

that means that the review was classified as GTA V, which is to be recorded as erroneous classification, and knowing which tags are more likely to be classified as other tags would have been very useful to represent similarities between different games. Now that this class is explained, the output of the class would be something like this:

The group A has the next accuracies for each other class:
B average cos distance: -0.08151683720294386
B has an accuracy of : 0/400
A average cos distance: 5.093872889119667E-4
A has an accuracy of : 0/400
C average cos distance: 0.7569797006249428
C has an accuracy of : 400/400
D average cos distance: 0.19814932759851217
D has an accuracy of : 0/400
E average cos distance: 0.5440101922303439
E has an accuracy of : 0/400
F average cos distance: 0.025434730636952736
F has an accuracy of : 0/400

The group B has the next accuracies for each other class:
B average cos distance: -0.037045918115909446
B has an accuracy of : 0/400
A average cos distance: -0.045483740557806414
A has an accuracy of : 0/400
C average cos distance: 0.7549091298040003
C has an accuracy of : 399/400
D average cos distance: 0.19445993211120366
D has an accuracy of : 0/400
E average cos distance: 0.5314711512625218
E has an accuracy of : 0/400
F average cos distance: 0.006557742160221096
F has an accuracy of : 1/400

The group F has the next accuracies for each other class:
B average cos distance: -0.04110614491859451
B has an accuracy of : 0/400
A average cos distance: -0.04103634840539598
A has an accuracy of : 0/400
C average cos distance: 0.7270043413882377
C has an accuracy of : 393/400
D average cos distance: 0.19797361596487462
D has an accuracy of : 0/400
E average cos distance: 0.4780905103310943
E has an accuracy of : 0/400
F average cos distance: 0.08513673244291567
F has an accuracy of : 7/400

This is an actual output of the testing program from the word2vec prototype. At first, it was thought to be a mistake from the reviews, however, after getting similar results in 5 different tries, with different random type A partitions (with different reviews in each of those 5 partitions) it was clear that there was an underlying problem with either the program or how it was getting executed. Yet another issue was submitted to the deeplearning4j github asking for help. For reference it can be found here:

https://github.com/deeplearning4j/deeplearning4j/issues/1657

The expert user raver119 offered his help once again to see what was going on with the dominancy of group C in the results. After 5 days of testing, skyping, and talking through Gitter, it was found that the whole text was very similar, or as he would put it: “Issue was investigated: actually single-domain corpus, with neglectible differences between labels, etc.”

After a last conversation with him through Gitter, he stated that the paravec software was not as yet as sharp as it should be, adding it to how similar the text was created a lot of problems. He recommended to use another example found on their Github. However, after a lot of weighting pros and cons of using word2vec, it was clear that the software may have been too new for stable testing. Mallet was selected as the next branch for testing and the word2vec branch of this project was closed.

3.3.3 Final Prototype: Mallet-based SA.

The Mallet Semantic analyzer was much easier to implement. Rather than using a lot of complex structures with few indications on how to properly set them up. Rather than using neural networks, that are more demanding both processing and time wise, Mallet uses hidden Markov Chains, maximum entropy Markov Models and Conditional Random fields, which can overall be summarized as LDA and LSA processing. Mallet included a set of tutorials and very concise slides that would explain how data was transformed and processed. Firstly, data is converted through a series of pipes to a numeric form, and from there it is processed depending on what the user programs. There is a wide array of models at the user’s disposition. The model that seemed to work best and, in the end, was the one used for the different models shown in this project, was LDA.

For this last part of the project, there were 3 classes. The ImporterSM class, the TopicModel class and the gameModel class. They follow a very similar structure to the one used for word2vec: the importer takes the data created by Module II and places it in a defined input, ready for the topic modeler to process it, afterwards, the model is tested and evaluated thanks to the gameModel class.

It is very important to reiterate that Mallet creates **topics** out of text. Essentially, it maps word appearances to a certain topic. So for example, if it were to be fed with sentences related to the soccer club Real Madrid, it would be able to estimate that once the words “Cristiano Ronaldo, Europe Cup, Casillas…” and others were found together on 100 comments out of 900, it would be able to tell that those 100 comments are talking about the same thing, but the computer would lack the human reasoning to be able to know that it is talking about soccer. This is very important to this part of the project, since what it means, is that Mallet will be able to map topics to a group of words, but not game titles, so the recommendation system would not be completed by just the topic modeling. In order to introduce this last package that makes the semantic analyzer, the classes in it, the different inputs, outputs and processes will be briefly introduced.

- **ImporterSm class**
  - **Input:** The formatted reviews in a single file per game. Only corpus (review text) and no numeric data.
  - **Processes:** The class places the data through a set of pipes that Mallet provides. More specifically, it applies the stop word list (removes the words from that list), it tokenizes data, and then turns it into a numeric form.
  - **Output:** Returns the information in an InstanceList (a class from the Mallet library that stores the corpus information).
- **TopicModel class**
  - **Input:** Takes in the instance list
  - **Processes:** It makes two threads to process data, sets the number of iterations to run the model through, and it processes the data through a ParallelTopicModel and a given number of topics (it can be adjusted).
  - **Output:** This builds a model based on topics, it creates topics, assigns them an id (this is important, it assigns each topic an id, not a game to a topic), and each topic is a probability density function. This function depends on the apparition of certain words. In a nutshell, it maps a set of words to a topic.
- **GameModel class.**
  - **Input:** takes in the model and the instance list from both classes, also, it takes in the "tags"(Appendix A) from the game list that is been tested with. These tags have been manually extracted from Steam's website. The model is given a list of topics, that after a text input returns a probability from 0 to 1, this probability states the chance of belonging to a topic.
  - **Processes:** Takes the list of tags per game, and processes it through the model. The highest probability from all the topics after taking the tag list returns the most probable topic due to that list of tags.
  - **Output:** A list of games mapped to a list of topics, together with an interactive program that allows user input and recommends a game based on what the user wrote.

This is done, so that if a user is talking about a game, not necessarily one that s/he knows of, will be able to find it. The point would be that the user could give a short, or long description of his desired game, and afterwards, the classifier will be able to evaluate the text and refer him/her to the most likely choice for this game based on what the other users have said.

In order to make the classifier more accurate, different partitions were tried out. First of all, reviews with cropped length and both positive and negative recommendations were tried out. This first model was clunky at best, providing successful hit rates for only about 30% of the reviews in the testing partitions, and very few successful interactive recommendations that actually pleased the testing users. The second model included every review, no matter the length (Preprocessing rule II was completely ignored) and the results were more catastrophic than before. About 20% of the models returned successful hit rates above than 50% for the testing reviews.

It was clear by that time that something was failing, and so, the models were carefully examined for a period of five days. During this time, it was discovered that there were two games that were highly problematic, first of all the RPG_Maker_V, which was not really a game, but a game making software. Therefore, since the corpus of this software was quite chaotic, removing this game, its reviews, and tags made the overall corpus more focused in the "gaming" domain and it reduced miss rates for the groups that it was assigned to. Also, "Youtuber's life" suffered from a similar problem and a very small review pool (only about 400 reviews, the games have an average review pool of 4,000 reviews), so it was also removed.

After these changes, the models were starting to make more sense, the topic count was also changed, and it seemed to reach the best accuracy at 20 topics for the now formally pool of 15 games.

The last thing that was changed before trying a new model, was changing the review pool, to only the positively rated reviews (the ones that actually recommended the game). This was done with the hypothesis in mind that when people are asking for a game, knowing what is good about the game is a lot more relevant than knowing what is bad. This seemed to be the case, since the models ended up successfully classifying reviews for 12/15 games on average after testing it 10 times. It seemed that the development was at its end, since a correct classifier was found.

It is also important to mention that in order to test the "recommendation" power of this software, a pool of over 20 gamers were personally introduced to the classifier and asked to try the software in interactive mode. The models were compiled and the user was asked to write a description of the desired game, rather than shutting down after saving the model, the program kept on running asking the user for descriptions and choosing a game or two games from the list. Afterwards, they were presented with a quality survey to evaluate the application and its recommendation quality. This survey is presented in the next section, but it is referred to here in order to justify the acceptance of the model by a small "hardcore-gaming" community.

4 Results

The main topic modeler was run 3 times, and yielded different results. Every result can be examined more closely in the appendices. There are 3 parts to every result:

1. **Topic creation:** This is the first step of the semantic processing. The whole data is condensed into a variable amount of topics that are represented by an id, which corresponds to a certain number of words appearing in the same text. In order to show these topics in a more comprehensive way, they will be shown as a series of the top 5 most common words for each topic, however, it should be noted that these are only the most probable words, not all of them.

2. **Mapping topics to games:** This process takes the tags taken in the Steam website as a voted list of word that defines a game and then asks the model which topic is most related to such list of words. The most related topic to such list is then considered to be the one talking about the game. This two-way mapping is needed to make sure that the user's opinion is taken into consideration in both steps.

3. **Classifying reviews:** The reviews, as explained before, are split into a training-test partition, this part of the process takes every game's test partition and asks the model for its dominant topic within the mapped domains in the previous step. To clarify, this means that if there are topics that are not assigned to a game in step 2, they will not be taken into consideration in this step.

4.1 Partition I:

Partition I: Topic summaries.

0	story world quests combat	10	rocket cars league friends
1	fun ive good time	11	gear boss action story
2	online story rockstar fun	12	total warhammer units battles
3	paradox stellaris empire strategy	13	amazing years year made
4	insurgency realistic team tactical	14	survival dinos early dinosaurs
5	battleborn fun borderlands moba	15	dont time people playing
6	competitive team skins fun	16	dead loot warhammer fun
7	multiplayer campaign fast fps	17	graphics optimization low kill
8	gear metal mgs kojima	18	keeper campaign bugs early
9	dark boss series bosses	19	story feel fps great

This information is for display purposes only, since without a deep knowledge of every game analyzed in this experiment, it is very unlikely that the reader will be able to make sense of the data. Afterwards, the program was fed with the tags of every game (see Appendix A) and every game was mapped to a group.

Partition I: Topic Mapping to games

Group **0** is mapped to the next games:

[DARK_SOULS_III, METAL_GEAR_SOLID_V_THE_PHANTOM_PAIN, The_Witcher_3_Wild_Hunt]

Group **18** is mapped to the next games:

[War_for_the_Overworld]

Group **2** is mapped to the next games:

[Grand_Theft_Auto_V]

Group **3** is mapped to the next games:

[Stellaris]

Group **4** is mapped to the next games:

[Insurgency, Counter_Strike]

Group **5** is mapped to the next games:

[Battleborn]

Group **7** is mapped to the next games:

[DOOM, Warhammer_Vermintide]

Group **10** is mapped to the next games:

[Rocket_League]

Group **11** is mapped to the next games:

[METAL_GEAR_RISING_REVENGEANCE]

Group **12** is mapped to the next games:

[Total_War_WARHAMMER]

Group **14** is mapped to the next games:

[ARK_Survival_Evolved]

This data is mostly straightforward for topics in which there is only one game mapped to. If there is more than one game mapped to a topic, it means that Mallet has identified a similarity in those games. In case of **Group 0**, the similarities are well known throughout steam community. First of all, they are all very complex games, which have RPG-like characteristics to them, they are all third person, and they are all open world. Lastly, all of these games are overwhelmingly positive classified on steam. The link that Mallet creates through the topic model is therefore justified. The second group that contains more than 1 game is **Group 4**, which is a group assigned to two very competitive first person shooters with a lot of realistic elements to them. Anyone playing both games can tell that they are very similar. **Group 4** is very important to this experiment, since it shows that game genres can be assigned dynamically and implicitly thanks to content (reviews) brought by the users. Lastly, **Group 7** puts together two shooters that are a bit different in game mechanics, Warhammer Vermintide is well known for having a lot of hardcore RPG mechanics to it, and DOOM has a set of abilities for the users to unlock but it could not be classified as an RPG game *per se*. The similarity in these two games lays on the fast action of each game, and on the setting. In both games, you shoot constantly fighting rounds of demons, rats and they are both Gore. From here on the links between the games within the same groups are justified, and from here on, the recommendation potential of the developed software will be examined.

Once the games were mapped, the test partition was run through the model, using a modified version of the result class to be adapted from cosine distance to average chance of belonging to a class. This information is displayed in Appendix E, since each model is composed of 15 tables with at least 12 rows to each table. In order to provide a condensed view, a summary will be provided. Firstly, every game review test partition for a single game carries the following information:

Table 4.1: War for the Overworld	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0075	0/93	0%	100%
DOOM, Warhammer_Vermintide	0.0049	0/93	0%	100%
Stellaris	0.0068	0/93	0%	100%
Total_War_WARHAMMER	0.0098	2/93	2%	98%
War_for_the_Overworld	0.2845	85/93	91%	9%
Rocket_League	0.0046	0/93	0%	100%
Insurgency, Counter_Strike	0.0077	3/93	3%	97%
ARK_Survival_Evolved	0.0046	1/93	1%	99%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.0031	0/93	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0020	0/93	0%	100%
Battleborn	0.0115	2/93	2%	98%

The title of the table shows which one of the game reviews test partition was under testing. To be clear, every review tested in Table 1.1 belongs to War for the Overword. What is under testing is the capability of the system to tell which reviews belong to which game. **P(B)** is short for Probability of belonging to the game topics (which are grouped at the leftmost column). The third column, shows how many of the test reviews were classified as the topic on its row. Lastly, the other two columns show the different hit and miss rate of the classifier. The column with most hit chance is highlighted blue if it is the correct game (as in this case), or orange if it is incorrect (see other examples Appendix C,D,E).

Mostly every game had its reviews correctly referred back to the group that it was mapped to. Here is a short summary of the data shown in the Appendix D.

Table 4.2: Successfully Mapped Games: Partition I		12/15
Game Title	**Hit rate**	**Group assigned**
War_for_the_Overworld	91%	War_for_the_Overworld
Battleborn	88%	Battleborn
The_Witcher_3	75%	DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3
ARK_Survival_Evolved	73%	ARK_Survival_Evolved
Rocket_League	77%	Rocket_League
Insurgency	87%	Insurgency, Counter_Strike
GTA_V	82%	GTA_V
Stellaris	80%	Stellaris
DOOM	85%	DOOM
Total_War_WARHAMMER	84%	Total_War_WARHAMMER
METAL_GEAR_RISING_REVENGEANCE	85%	METAL_GEAR_RISING_REVENGEANCE
Counter_Strike	29%	Insurgency, Counter_Strike

As it can be seen, every game but Counter Strike has a very positive hit rate, not going under 70% at any point of the review process. This is a very positive result, since it shows than in 11 out of 15 cases, this software is able to identify the topics of the games, and

point users to a group based on a game description. However, not every result is positive. The next games have wrongly mapped groups.

Table 4.3: Unsuccessfully Mapped Games: Partition I		3/15
Game Title	**Hit rate**	**Group assigned**
DARK_SOULS_III	26%	War_for_the_Overworld
Warhammer_Vermintide	25%	War_for_the_Overworld
METAL_GEAR_SOLID_V	23%	War_for_the_Overworld

It is very hard to find why the system is not working correctly in these cases. However, rather than just discarding this information as outliers or statistical anomalies, these failures will be tried to be justified. As it can be seen, all of the games are wrongly mapped to the War for the Overworld game, which is extremely odd, since the games have nothing to do with War for the Overworld. WfO (abbreviation for the game title) is a dungeon manager game, with a top down perspective. It only shares a bit of environment with Dark Souls III and Warhammer Vermintide since they are all quite gothic and gore in nature, they also share RPG elements. However, Metal Gear Solid V the Phantom Pain is a very odd match. It is also shown that both Darksouls III and Metal Gear Solid V belong to the same group (group 0). It may be that the other game within that group, Witcher III, is a lot more dominant. It is also worth noticing that Witcher III users are a lot more mature (in writing) than the users from Dark Souls 3 and Metal Gear V. This is an assumption made due to the intimate knowledge of the gamming community, but it can be easily seen by looking at more than 200 reviews from every game and comparing the wording and structure of the reviews. This is not a formal fact, but more of a supposition coming from the expertise in the field.

4.2 Partition II:

This is a new partition, with a testing split that differs greatly from the one used before. Once again, these partitions are created randomly selecting from a 90%-10% distribution.

Partition II: Topic summaries

0 dungeon keeper campaign bugs original
1 good great time dont gameplay
2 space paradox stellaris empire strategy
3 metal gear mgs kojima missions
4 csgo competitive team skins counterstrike
5 server survival dinos early dinosaurs
6 fun playing people players lot
7 people time dont playing hours
8 witcher world quests combat rpg
9 souls dark boss series bosses
10 ive feel feels time review
11 war total warhammer units campaign
12 characters battleborn moba borderlands pvp
13 left dead loot warhammer rats
14 story feel graphics series character
15 metal gear boss action fun
16 fun rocket cars league friends
17 doom multiplayer fps campaign fast
18 fps realistic insurgency team tactical
19 gta online rockstar fun friends

Partition II: Topic Mapping to games

Group **0** is mapped to the next games:

[War_for_the_Overworld]

Group **16** is mapped to the next games:

[Rocket_League]

Group **17** is mapped to the next games:

[DOOM]

Group **18** is mapped to the next games:

[Insurgency, Warhammer_Vermintide, Counter_Strike]

Group **2** is mapped to the next games:

[Stellaris]

Group **3** is mapped to the next games:

[METAL_GEAR_SOLID_V_THE_PHANTOM_PAIN]

Group **19** is mapped to the next games:

[Grand_Theft_Auto_V]

Group **5** is mapped to the next games:

[ARK_Survival_Evolved]

Group **8** is mapped to the next games:

[DARK_SOULS_III, The_Witcher_3_Wild_Hunt]

Group **11** is mapped to the next games:

[Total_War_WARHAMMER]

Group **12** is mapped to the next games:

[Battleborn]

Group **15** is mapped to the next games:

[METAL_GEAR_RISING_REVENGEANCE]

This new group from the other random partition created some very interesting results. As before, there are groups that contain more than one game, but this time there are only two groups such as this. **Group 18**, together with Insurgency and Counter Strike, also contains Warhammer Vermintide. This can be easily justified by the fact that all of these three games are the only cooperative competitive shooters that are played within small environments (maps or arenas) over and over again. Also, the group that last time contained Dark Souls III, Witcher III and Metal Gear V, now only contains the first two. This will show to be a very positive change when it comes to classify these games.

Table 4.4: Successfully Mapped Games: Partition II		**13/15**
Game Title	**Hit rate**	**Group assigned**
War_for_the_Overworld	97%	War_for_the_Overworld
Battleborn	83%	Battleborn
The_Witcher_3	73%	DARK_SOULS_III, The_Witcher_3
ARK_Survival_Evolved	75%	ARK_Survival_Evolved
Rocket_League	73%	Rocket_League
Insurgency	85%	Insurgency, Counter_Strike
GTA_V	80%	GTA_V
Stellaris	81%	Stellaris
DOOM	85%	DOOM
Total_War_WARHAMMER	81%	Total_War_WARHAMMER
METAL_GEAR_RISING_REVENGEANCE	77%	METAL_GEAR_RISING_REVENGEANCE
Counter_Strike	29%	Insurgency, Counter_Strike
METAL_GEAR_SOLID_V	78%	METAL_GEAR_SOLID_V

Table 4.5: Unsuccessfully Mapped Games: Partition II		2/15
Game Title	**Hit rate**	**Group assigned**
DARK_SOULS_III	33%	War_for_the_Overworld
Warhammer_Vermintide	24%	War_for_the_Overworld

As it can be seen, classifying Metal Gear V the Phantom Pain as its own game, produced a positive classifying of almost 80% of the reviews. Which is obviously a very good increment from the previous classifier's accuracy. In the comparative study, the data will be compared more closely. At this point, it can be said that the new classifier's choice was a lot more accurate than the one before.

4.3 Partition III:

As a new partition is created, the same data as the other partitions will be displayed.

Partition III: Topic summaries

0 people fun dont playing back
1 gta online rockstar story friends
2 series end man weapon year
3 csgo competitive team skins players
4 time great feel ive good
5 story amazing beautiful world ive
6 witcher story world quests combat
7 server survival dinos early dinosaurs
8 characters battleborn fun borderlands pvp
9 space paradox stellaris empire strategy
10 souls dark boss series bosses
11 war total warhammer units battles
12 doom multiplayer fps campaign fast
13 metal gear boss story action
14 good great dont fun buy
15 dungeon keeper campaign bugs original
16 story metal gear mgs kojima
17 left dead loot warhammer rats
18 fun rocket cars league friends
19 fps realistic insurgency team tactical

Partition III: Topic Mapping to games

Group **16** is mapped to the next games:

[METAL_GEAR_SOLID_V_THE_PHANTOM_PAIN]

Group **1** is mapped to the next games:

[Grand_Theft_Auto_V]

Group **18** is mapped to the next games:

[Rocket_League]

Group **19** is mapped to the next games:

[Insurgency, Counter_Strike]

Group **6** is mapped to the next games:

[DARK_SOULS_III, The_Witcher_3_Wild_Hunt]

Group **7** is mapped to the next games:

[ARK_Survival_Evolved]

Group **8** is mapped to the next games:

[Battleborn]

Group **9** is mapped to the next games:

[Stellaris]

Group **11** is mapped to the next games:

[Total_War_WARHAMMER]

Group **12** is mapped to the next games:

[DOOM, Warhammer_Vermintide]

Group **13** is mapped to the next games:

[METAL_GEAR_RISING_REVENGEANCE]

Group **15** is mapped to the next games:

[War_for_the_Overworld]

This new partition corrects only one mapping from the previous partitions: it takes Warhammer Vermintide back together with DOOM. This would originally make the reader suppose that the impact to this partition will be minimum, and that the changes will be very insignificant compared to the other partitions. This is however, not the case.

Table 4.6: Successfully Mapped Games: Partition III		**12/15**
Game Title	**Hit rate**	**Group assigned**
War_for_the_Overworld	91%	War_for_the_Overworld
Battleborn	85%	Battleborn
The_Witcher_3	65%	DARK_SOULS_III, The_Witcher_3
ARK_Survival_Evolved	77%	ARK_Survival_Evolved
Rocket_League	73%	Rocket_League
Insurgency	84%	Insurgency, Counter_Strike
GTA_V	85%	GTA_V
Stellaris	86%	Stellaris
DOOM	87%	DOOM
Total_War_WARHAMMER	85%	Total_War_WARHAMMER
METAL_GEAR_RISING_REVENGEANCE	81%	METAL_GEAR_RISING_REVENGEANCE
METAL_GEAR_SOLID_V	72%	METAL_GEAR_SOLID_V

Table 4.7: Unsuccessfully Mapped Games: Partition III		**3/15**
Game Title	**Hit rate**	**Group assigned**
DARK_SOULS_III	40%	GTA_V
Warhammer_Vermintide	30%	GTA_V
Counter_Strike	35%	GTA_V

As it is shown, the whole unsuccessfully mapped games, now belong to GTA rather than to War for the Overworld. This is somewhat a positive change when looked at from a gamer's perspective. First of all, every game has shooting elements, same as GTA V. Every game has skills defined and a class system. Most importantly, but less easily seen, is that all of these four games have something very relevant in common: a "Raging community". This is a somewhat hard concept to formally explain, and it taps entirely into online anthropology. These four games depend on the interaction with other users in order to win or lose, this in the long term causes a lot of grudges, and an overall more toxic environment. The perfect example to support this claim is the community of League of Legends, the most played online game at this moment, that has the two conditions mentioned before, interaction with others users for both winning and losing conditions. [16]

[16] https://www.reddit.com/r/MMORPG/comments/2ged17/what_game_has_the_worst_community/ last accessed 28/06/2016

This could be the cause of the overall feeling of the reviews and the inaccuracy of the classifier. This, once again, is an opinion which has tried to be defended, but this is not proven in any way.

4.4 Comparative Study and reflections.

Table 4.8: Successfully Mapped Games: Overall			
Game Title	**Hit rate Partition I**	**Hit rate Partition II**	**Hit Rate Partition III**
War_for_the_Overworld	91%	97%	91%
Battleborn	88%	83%	85%
The_Witcher_3	75%	73%	65%
ARK_Survival_Evolved	73%	75%	77%
Rocket_League	77%	73%	73%
Insurgency	87%	85%	84%
GTA_V	82%	80%	85%
Stellaris	80%	81%	86%
DOOM	85%	85%	87%
Total_War_WARHAMMER	84%	81%	85%
METAL_GEAR_RISING REVENGEANCE	85%	77%	81%
Counter_Strike	29%	29%	----------
METAL_GEAR_SOLID_V	----------	78%	91%
OVERALL	72%	76%	75%

When looking at the comparative study, it is clear that, on average, the second partition is the best classifier. It distributes the probability more evenly and it will be a better overall classifier. The one that contains more maximum hit rates is the first classifier, but it does not compensate the larger spread with respect to the other classifiers.

Referring back to the results shown, it is clear that overall, the classifiers have a fair performance, correctly classifying 12 out of 15 games at the worst, and 13 out of 15 at its best. These classifiers have been the outcome of an enormous amount of work put into carefully tuning the classifiers, to find the best games, best parameters and best partition methods. It is clear by the numbers that they serve its purpose well.

5 Future work and conclusions

This project has yielded some very interesting results. It would seem clear that it classifies some games better than others, and hence it brings the question whether recommendations systems should or should not be based in the average user review, which is essentially what has been done here. The problem is not one of processing power, or data size, but it lays within the incredibly informal and humor based internet society. Memes, sarcasm, ironies, ASCII art, and many other particularities of the corpus belonging to these kind of communities make them extremely hard to analyze and interpret via Machine Learning.

It has been a difficult project. The crawler was extremely time consuming, and certain reasonable upgrades, such as the use of a database, and the removal of the text files could have been implemented if not for the long development time that the first part of the project had. It is clear that when dealing with online systems that are not 100% static (very rare occurrence nowadays), projects such as these are subject to a lot of problems due to upgrades, API access changes, and how easy the host makes it for the developer to freely extract data from its services. These problems usually have basic solutions, but being a bachelor thesis made it more challenging. The second module was trivial and the hard courses that excelled at teaching Abstract Data Types and Object Oriented Programming made it very easygoing to program the different structures that supported this large project. Lastly, looking for new technologies to use in this project was not trivial. The absence of a deep understanding that only comes from years of dedication to a field such as semantic analysis has been noted, maybe with more time to dedicate to the semantic field, a broader spectrum of results could have been created. However, this project has not been a failure at all.

The amount of learning that has been achieved in this project is overwhelming, since it is a very broad project in terms of techniques and disciplines. Essential parts of what makes a good Computer Engineer have been improved. Code commenting, problem solving, working with outside teams to manage issues, Java programming, statistics and probabilities, algorithm design and many more. There has been new knowledge that has been tapped into, such as understanding how the World Wide Web works, semantic analysis, statistical processing, neural nets, text processing algorithms, stop words usage, LDA, LSA, the use of labels when training test and more specifically, the use of technologies such as Mallet and Word2Vec.

Its success not only lies in the long path to its end, and how much it has been learnt thanks to it. But the project itself has been a success. By the end, three classifiers were trained, each one with its strengths and weaknesses, but each one ready to recommend a group of games, or just one based only in user description and information. This is a field that is new to the development team, but it seems from what has been researched, that it is a relatively new approach to recommending systems, that may or may not be successful in the future.

It has not been perfect however. Future improvements on this project will surely include formalizing data treatment by the implementation of a SQL driver that will manage the information, rather than pass it along through text files. Also, taking in a greater number of games and opening the project to more information would surely bring forward new results that could serve to improve the original TopicModel class. The creation of a website with this project so that other users can evaluate its effectivity would also be a great stepping stone for this project. Lastly, as explained in the comparative study of Chapter 5, combining classifiers to be able to take the strengths of the different models and dispose of its weakest parts is something to consider. In order to do all of this however, the crawler itself should be improved to become more efficient and mine with different threads.

When taking a last look at the comparative study of Chapter 4, it is clear that each classifier is better at classifying certain games than the others, each is better than the other in some aspects, so to say. This would bring an interesting argument about which one is the “best” classifier or recommender. Scientifically, the one that maximizes overall approval, in this case classifier 2, could possibly be the “best” classifier, since it makes sure that the system has the highest hit rate. However, it may be more interesting building a system that could take a lot of classifiers and consult one or another depending on their specialization. For example, an overall classifier could be used to direct the user to a group, and then, check a classifier that better recommends a certain group of games could be consulted in order to bring a sharper system. However, it would be quite complex. There could be a broad range of classifiers, each to recommend a genre, with its own parameters (as explained in the last part of Chapter 4, each classifier shown here is trained under the same parameters) that would, for example, be trained under a smaller max topic parameter, which would force the classifier to be more general, or with a higher max topic, that could potentially yield more specific results. Also the classifiers could be trained with different portions of the data, for example, they could take in reviews only for the “shooter” genre, so as to be able to distinguish better between each specific shooter.

The last thing that will be brought up in this report will be the usages of this technology. First of all, Steam could use its base to improve the recommendation system that it currently has. Secondly, this very same algorithm can be used to predict if comments actually relate to the general subject of a video, a movie or a game. Thirdly this recommendation system could be expanded to anything that could yield this amount of comments, not Amazon, restaurants, or other sale based websites at its current place, since the number of comments and product reviews are not that high, but surely there has to be another topic that has the same amount of passion as this.

This project has been a great journey, it has brought a lot of stress, a lot of learning, and the satisfaction of creating something new and unique. Its hardships have been many, but the learning that has brought with it has made it a very valuable experience.

Glossary

API	Application Programming Interface
Bag of words	It is a model which is characterized by a simplification of the representation used in natural language processing. This technique considers text as a "bag of words" and takes out grammar and word order, but keeping a tally of the words used.
Casual game	It is said that a game is a casual one when it lacks a competitive structure and is of a more easygoing nature.
Competitive game	It is said that a game is competitive when it has a ranking system that matches players with the same skill level. It also implies a certain degree of e-sports
Crawler	A system that is capable to move through a web structure, extracting and collecting information.
E-sports	Games that are considered to be a digital sport, such as Counter Strike, League of Legends, Defense Of The Ancients 2 .
First Person	Game perspective in which the user is only shown the hands of the character s/he is controlling.
IP ban	Security measure used by servers to deny DDoS attacks, it simply blocks petitions from a set of IP addresses or an IP address range.
JS	JavaScript.
LDA	Latent Dirichlet allocation, it is a generative statistical model used for topic discovery in some semantic analysis.
LSA	Latent Semantic Analysis, it is a technique used in natural language processing based on discovering text concepts and operating with a matrix of such concepts to discover similarities between different texts.
Mature	Game genre that implies adult content, such as gore, sexual intercourse or nudes.
Multiplayer	A game genre that involves matching players with or against each other to reach a determined goal.
Open World	Game genre or trait that implies that the user of the game is left on an open area that s/he is free to explore in whichever way the user wants.
PvP	Player versus Player, which is a type of game in which users are matched against each other.
REST	Representational State Transfer
RPG	Role Playing Game, a game that involves the use of a character in a fictional setting, leveling up, and character customization.
SA	Semantic Analyzer.
Shooter	A common game genre based on the usage of guns as the main weapon of the game
Third Person	Game perspective in which the user is shown the full body of the character that s/he is controlling.
W2V	Word2Vec
Web Scrapping	Technique used to collect data from HTML documents.

Appendices

A Game tags list:

ARK_Survival_Evolved
Early Access Survival Dinosaurs Open World Multiplayer Crafting Building Adventure Co-op Action First-Person Base-Building Sandbox Massively Multiplayer Singleplayer RPG Dragons Sci-fi MMORPG Indie

METAL_GEAR_SOLID_V_THE_PHANTOM_PAIN
Stealth Open World Story Rich Action Tactical Cinematic Great Soundtrack Third Person Singleplayer Atmospheric Horses Adventure Multiplayer Third-Person Shooter Replay Value Sandbox Shooter Dark Sci-fi Heist

The_Witcher_3_Wild_Hunt
RPG Open World Story Rich Atmospheric Mature Fantasy Adventure Choices Matter Third Person Singleplayer Action Great Soundtrack Nudity Medieval Dark Fantasy Multiple Endings Magic Action RPG Dark Sandbox

DOOM
FPS Action Gore Demons Shooter First-Person Multiplayer Sci-fi Classic Horror Singleplayer Fast-Paced Great Soundtrack Atmospheric Difficult Remake Zombies Survival Horror Co-op Memes

DARK_SOULS_III
Dark Fantasy Difficult Atmospheric RPG Lore-Rich Third Person Exploration PvP Story Rich Co-op Action RPG Adventure Open World Action Multiplayer Great Soundtrack Singleplayer Psychological Horror Nudity Funny

Grand_Theft_Auto_V
Open World Action Multiplayer First-Person Third Person Crime Adventure Shooter Third-Person Shooter Singleplayer Atmospheric Mature Racing Sandbox Co-op Great Soundtrack Funny Comedy Moddable RPG

Insurgency
FPS Realistic Tactical Multiplayer Action Shooter Military Team-Based Strategy Co-op Indie First-Person Competitive Online Co-Op War Simulation PvP Atmospheric Singleplayer Adventure

Battleborn
FPS Action Multiplayer MOBA Co-op Shooter First-Person Comedy Singleplayer PvP Sci-fi Class-Based Funny Action RPG Team-Based Online Co-Op Memes RPG Space Casual

Stellaris
Space Strategy Grand Strategy Sci-fi 4X Real-Time with Pause Exploration Multiplayer Singleplayer RTS Simulation Replay Value Great Soundtrack Sandbox Atmospheric Moddable Story Rich Open World

Warhammer_Vermintide
Action Online Co-Op Co-op Dark Fantasy Gore Games Workshop Multiplayer FPS First-Person Action RPG Atmospheric Fantasy Survival Indie Hack and Slash Horror RPG Adventure Female Protagonist Singleplayer

METAL_GEAR_RISING_REVENGEANCE
Great Soundtrack Action Hack and Slash Spectacle fighter Swordplay Character Action Game Third Person Singleplayer Cyberpunk Ninja Fast-Paced Gore Sci-fi Replay Value Adventure Difficult Mechs Story Rich Stealth Beat 'em up

Total_War_WARHAMMER
Strategy Fantasy RTS Games Workshop War Turn-Based Strategy Multiplayer Grand Strategy Dark Fantasy Action Tactical Atmospheric Singleplayer Turn-Based Story Rich Co-op Gore Open World RPG Warhammer 40K

Counter_Strike
FPS Multiplayer Shooter Action Team-Based Competitive Tactical First-Person e-sports PvP Online Co-Op Military Co-op Strategy War Trading Realistic Difficult Fast-Paced Moddable

Rocket_League
Racing Multiplayer Soccer Sports Competitive Team-Based Football Online Co-Op Action Co-op Funny Fast-Paced Local Multiplayer Local Co-Op Great Soundtrack Split Screen 4 Player Local Singleplayer Indie Casual

War_for_the_Overworld
Strategy Indie RTS God Game Fantasy Kickstarter Sandbox Singleplayer Management Villain Protagonist Dungeon Crawler Funny Multiplayer Simulation Base-Building

B Tables 1: Accuracies for every game in partition 1.

Table 1.1: War for the Overworld	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.007585400557057288	0/93	0%	100%
DOOM, Warhammer_Vermintide	0.004924070924268071	0/93	0%	100%
Stellaris	0.006845474316855438	0/93	0%	100%
Total_War_WARHAMMER	0.009801592456843658	2/93	2%	98%
War_for_the_Overworld	0.28452104460237587	85/93	91%	9%
Rocket_League	0. 004659259639611257	0/93	0%	100%
Insurgency, Counter_Strike	0.007769376585388766	3/93	3%	97%
ARK_Survival_Evolved	0.004561350290858301	1/93	1%	99%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.003082186429845914	0/93	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.002053129193490111	0/93	0%	100%
Battleborn	0.011557036747237738	2/93	2%	98%

Table 1.2: Battleborn	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.011472872202340773	6/227	2%	88%
DOOM, Warhammer_Vermintide	0.0046276330392499496	2/227	1%	99%
Stellaris	0.005078708708589238	0/227	0%	100%
Total_War_WARHAMMER	0.006206332754796953	2/227	1%	99%
War_for_the_Overworld	0.00886726308250212	4/227	2%	98%
Rocket_League	0.006669945665581839	4/227	2%	98%
Insurgency, Counter_Strike	0.011721156660981941	4/227	2%	98%
ARK_Survival_Evolved	0.002891899882923031	1/227	0.5%	99.5%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.0029759805034597683	0/227	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.004956570210505033	5/227	2%	98%
Battleborn	0.31389857452537895	200/227	88%	12%

Table 1.3: Witcher_3	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.012978096783094544	24/644	3%	97%
DOOM, Warhammer_Vermintide	0.004916029114176923	5/644	1%	99%
Stellaris	0.006543158958912284	11/644	2%	98%
Total_War_WARHAMMER	0.008677977039394685	21/644	3%	97%
War_for_the_Overworld	0.010143913128071223	52/644	8%	92%
Rocket_League	0.005059563401076073	12/644	2%	98%
Insurgency, Counter_Strike	0.008004706918829693	12/644	2%	98%
ARK_Survival_Evolved	0.0014141186980278692	1/644	0%	100%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.2228708035214046	487/644	75%	25%
METAL_GEAR_RISING_REVENGEANCE	0.006164975475974835	8/644	1%	99%
Battleborn	0.006519168431369719	11/644	2%	98%

Table 1.4: ARK_Survival_Evolved	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.015282618109083277	37/523	7%	93%
DOOM, Warhammer_Vermintide	0.004722396156249555	4/523	1%	99%
Stellaris	0.006429031417893845	5/523	1%	99%
Total_War_WARHAMMER	0.00706331461578811	5/523	1%	99%
War_for_the_Overworld	0.015163328503587052	51/523	10%	90%
Rocket_League	0.005875027488655491	12/523	2%	98%
Insurgency, Counter_Strike	0.008242017639233249	11/523	2%	98%
ARK_Survival_Evolved	0.2253088623404686	381/523	73%	27%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.0038744427589049273	4/523	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.003992099417902114	3/523	1%	99%
Battleborn	0.00849267595516125	10/523	2%	98%

Table 1.5: Rocket_League	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.01591773396370788	29/562	5%	95%
DOOM, Warhammer_Vermintide	0.00684587056169445	7/562	1%	99%
Stellaris	0.006755119607252276	6/562	1%	99%
Total_War_WARHAMMER	0.008239351428653929	12/562	2%	98%
War_for_the_Overworld	0.01051177278819627	42/562	7%	93%
Rocket_League	0.258431378589074	430/562	77%	23%
Insurgency, Counter_Strike	0.011276428133504535	19/562	3%	97%
ARK_Survival_Evolved	0.001471684737568457	2/562	0.5%	99.5%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.0027957185546593805	0/562	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0051559870346535456	3/562	0.5%	99.5%
Battleborn	0.009148355987326209	12/562	2%	98%

Table 1.6: Insurgency	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.011139359552082565	20/663	3%	97%
DOOM, Warhammer_Vermintide	0.007322071145940012	7/663	1%	99%
Stellaris	0.005874799106440028	3/663	0.5%	99.5%
Total_War_WARHAMMER	0.008218537101212422	7/663	1%	99%
War_for_the_Overworld	0.010779572182505446	28/663	4%	96%
Rocket_League	0.008014912212014981	8/663	1%	99%
Insurgency, Counter_Strike	0.30989796443502027	574/663	87%	13%
ARK_Survival_Evolved	0.002697662958749167	3/663	0.5%	99.5%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.0037489433059771197	2/663	0.5%	99.5%
METAL_GEAR_RISING_REVENGEANCE	0.004283162834158655	3/663	0.5%	99.5%
Battleborn	0.009807544262197737	8/663	1%	99%

Table 1.7: GTA_V	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.23876270225966165	302/376	82%	18%
DOOM, Warhammer_Vermintide	0.006292809269235388	2/376	0.5%	99.5%
Stellaris	0.005648971187032399	0/376	0%	100%
Total_War_WARHAMMER	0.011310128366482684	9/376	2%	98%
War_for_the_Overworld	0.015423813345866693	38/376	10%	90%
Rocket_League	0.010122406756854413	6/376	1%	99%
Insurgency, Counter_Strike	0.012404136836066957	8/376	1%	99%
ARK_Survival_Evolved	0.0023740273559351038	1/376	0%	100%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.007247116664476501	3/376	0.5%	99.5%
METAL_GEAR_RISING_REVENGEANCE	0.004794774655468415	3/376	0.5%	99.5%
Battleborn	0.008203929778711067	4/376	0.5%	99.5%

Table 1.8: Stellaris	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.009602708208192826	7/273	2%	98%
DOOM, Warhammer_Vermintide	0.0040311426384515275	3/273	1%	99%
Stellaris	0.2823264761475659	219/273	80%	20%
Total_War_WARHAMMER	0.013998918873228232	11/273	3%	97%
War_for_the_Overworld	0.010124915746691047	17/273	6%	94%
Rocket_League	0.006786590018307933	9/273	3%	97%
Insurgency, Counter_Strike	0.006741051109973604	3/273	1%	99%
ARK_Survival_Evolved	0.0016616261489169701	0/273	0%	100%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.0032439953665773635	1/273	0.5%	99.5%
METAL_GEAR_RISING_REVENGEANCE	0.0036133175323913185	0/273	0%	100%
Battleborn	0.0057752860085125545	3/273	1%	99%

Table 1.9: DOOM	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.01164465587286052	7/585	1%	99%
DOOM, Warhammer_Vermintide	0.2662121202861144	500/585	85%	15%
Stellaris	0.005045771168706361	1/585	0.5%	99.5%
Total_War_WARHAMMER	0.008611813220749585	9/585	1%	99%
War_for_the_Overworld	0.010894660854893658	23/585	4%	96%
Rocket_League	0.005060215664375808	6/585	1%	99%
Insurgency, Counter_Strike	0.014389909937849756	18/585	3%	97%
ARK_Survival_Evolved	9.907273645369094E-4	1/585	0.5%	99.5%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.004859211128674966	4/585	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.005783887343512537	7/585	1%	99%
Battleborn	0.007655040671474714 5	9/585	1%	99%

Table 1.10: Total_War_WARHAMMER	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.011232031197913323	10/321	3%	97%
DOOM, Warhammer_Vermintide	0.00515910849884074	1/321	0.5%	99.5%
Stellaris	0.008998889361962269	5/321	1.5%	98.5%
Total_War_WARHAMMER	0.297325419300726	271/321	84%	26%
War_for_the_Overworld	0.01160998304090544	19/321	6%	94%
Rocket_League	0.004025178198556828	3/321	0.5%	99.5%
Insurgency, Counter_Strike	0.006882914021896591	2/321	0.5%	99.5%
ARK_Survival_Evolved	0.0017495349317269003	1/321	0.5%	99.5%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.0036367604451574436	3/321	0.5%	99.5%
METAL_GEAR_RISING_REVENGEANCE	0.00546977045959523	2/321	0.5%	99.5%
Battleborn	0.007666627734898679	4/321	1%	99%

Table 1.11: Warhammer_Vermintide	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.011849357838544378	27/327	8%	92%
DOOM, Warhammer_Vermintide	0.007459858086373419	15/327	4%	96%
Stellaris	0.005957922594972605	16/327	4%	96%
Total_War_WARHAMMER	0.016504855908924948	42/327	12%	88%
War_for_the_Overworld	0.01142309177507012	83/327	25%	75%
Rocket_League	0.00604515265153152	11/327	3%	97%
Insurgency, Counter_Strike	0.015163814004102398	43/327	12%	88%
ARK_Survival_Evolved	0.0017281192844108575	5/327	1%	99%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.006412003049427793	20/327	7%	93%
METAL_GEAR_RISING_REVENGEANCE	0.006014714448489843	10/327	1.5%	98.5%
Battleborn	0.01628535120416342	55/327	17%	83%

Table 1.12: DARK_SOULS_III	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.014873771258798641	68/489	14%	86%
DOOM, Warhammer_Vermintide	0.004919880266670803	16/489	3%	97%
Stellaris	0.0055465998802126815	22/489	4%	96%
Total_War_WARHAMMER	0.01090119665454564	55/489	11%	89%
War_for_the_Overworld	0.009583290821975989	128/489	26%	74%
Rocket_League	0.00464889139896094	22/489	4%	96%
Insurgency, Counter_Strike	0.008592626290380444	46/489	8%	92%
ARK_Survival_Evolved	0.002049468423263892	7/489	1.5%	98.5%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.006666578264320777	40/489	8%	92%
METAL_GEAR_RISING_REVENGEANCE	0.009560235063808029	44/489	9%	91%
Battleborn	0.007675676238710422	41/489	8%	92%

Table 1.13: METAL_GEAR_RISING_REVENGEANCE	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.012236792832138493	9/398	2%	98%
DOOM, Warhammer_Vermintide	0.005007022751134318	3/398	1%	99%
Stellaris	0.006357901627951134	2/398	1%	99%
Total_War_WARHAMMER	0.007742151904571156	5/398	1%	99%
War_for_the_Overworld	0.00803984501580179	18/398	4%	96%
Rocket_League	0.00470532372384033	3/398	1%	99%
Insurgency, Counter_Strike	0.008220399024422098	7/398	2%	98%
ARK_Survival_Evolved	0.001605599906342483	3/398	1%	99%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.005477725118100397	4/398	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.29768343809397435	339/398	85%	15%
Battleborn	0.00825798449403805	5/398	1%	99%

Table 1.14: METAL_GEAR_SOLID_V	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0133021077065471	77/493	16%	84%
DOOM, Warhammer_Vermintide	0.005478989704696575	21/493	4%	96%
Stellaris	0.006466242484873808	25/493	5%	95%
Total_War_WARHAMMER	0.00837355076487079	43/493	8%	92%
War_for_the_Overworld	0.009225959729886915	114/493	23%	73%
Rocket_League	0.004717162372880044	23/493	4%	96%
Insurgency, Counter_Strike	0.012241245835352929	52/493	10%	90%
ARK_Survival_Evolved	0.0015047399533882001	4/493	1%	99%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.00880023124139574	32/493	7%	93%
METAL_GEAR_RISING_REVENGEANCE	0.015435836357437093	69/493	14%	86%
Battleborn	0.007128197086277641	33/493	7%	93%

Table 1.15: Counter_Strike	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.018062621609523943	68/428	16%	1484%
DOOM, Warhammer_Vermintide	0.007187740412342022	15/428	4%	96%
Stellaris	0.007939325278330108	18/428	4%	96%
Total_War_WARHAMMER	0.009544507208314299	25/428	5%	95%
War_for_the_Overworld	0.011253789006888313	97/428	22%	78%
Rocket_League	0.012917866473168373	29/428	6%	94%
Insurgency, Counter_Strike	0.04822585463063473	123/428	29%	71%
ARK_Survival_Evolved	0.0018420174415562293	3/428	1%	99%
DARK_SOULS_III, METAL_GEAR_SOLID_V, The_Witcher_3	0.004279430531055203	14/428	4%	96%
METAL_GEAR_RISING_REVENGEANCE	0.00455723171243323	7/428	2%	98%
Battleborn	0.010538669600700495	29/428	6%	94%

C Tables 2: Accuracies for every game in partition 2.

Table 2.1: War for the Overworld	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0103	1/93	1%	99%
METAL_GEAR_SOLID_V	0.0043	0/93	0%	100%
War_for_the_Overworld	0.2929	91/93	98%	2%
Stellaris	0.0055	0/93	0%	100%
Total_War_WARHAMMER	0.0064	0/93	0%	100%
DOOM	0.0039	0/93	0%	100%
Rocket_League	0.0027	0/93	0%	100%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0074	0/93	0%	100%
ARK_Survival_Evolved	0.0042	0/93	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0034	0/93	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0011	0/93	0%	100%
Battleborn	0.0050	1/93	1%	99%

Table 2.2: Battleborn	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0109	5/227	2%	98%
METAL_GEAR_SOLID_V	0.0035	3/227	1%	99%
War_for_the_Overworld	0.0084	12/227	5%	95%
Stellaris	0.0045	2/227	1%	1%
Total_War_WARHAMMER	0.0029	1/227	1%	1%
DOOM	0.0034	1/227	1%	1%
Rocket_League	0.0064	6/227	2%	98%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0093	4/227	2%	98%
ARK_Survival_Evolved	0.0015	1/227	1%	1%
DARK_SOULS_III, The_Witcher_3	0.0039	2/227	1%	1%
METAL_GEAR_RISING_REVENGEANCE	0.0051	1/227	1%	1%
Battleborn	0.2608	189/227	83%	17%

Table 2.3: The_Witcher_3_Wild_Hunt	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0123	35/644	5%	95%
METAL_GEAR_SOLID_V	0.0048	7/644	1%	99%
War_for_the_Overworld	0.0098	71/644	10%	90%
Stellaris	0.0050	4/644	1%	99%
Total_War_WARHAMMER	0.0041	7/644	1%	99%
DOOM	0.0042	6/644	1%	99%
Rocket_League	0.0051	11/644	1%	99%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0078	19/644	3%	97%
ARK_Survival_Evolved	0.0021	4/644	1%	99%
DARK_SOULS_III, The_Witcher_3	0.2026	474/644	73%	27%
METAL_GEAR_RISING_REVENGEANCE	0.0029	3/644	1%	99%
Battleborn	0.0026	3/644	1%	99%

Table 2.4: ARK_Survival_Evolved	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0163	33/523	6%	94%
METAL_GEAR_SOLID_V	0.0033	2/523	0%	100%
War_for_the_Overworld	0.0127	49/523	9%	91%
Stellaris	0.0064	9/523	2%	98%
Total_War_WARHAMMER	0.0041	8/523	2%	98%
DOOM	0.0037	4/523	1%	99%
Rocket_League	0.0049	6/523	2%	98%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0069	10/523	2%	98%
ARK_Survival_Evolved	0.2690	394/523	75%	25%
DARK_SOULS_III, The_Witcher_3	0.0046	3/523	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0018	2/523	0%	100%
Battleborn	0.0030	3/523	0%	100%

Table 2.5: Rocket_League	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0159	32/562	6%	94%
METAL_GEAR_SOLID_V	0.0048	3/562	0%	100%
War_for_the_Overworld	0.0101	63/562	12%	88%
Stellaris	0.0051	0/562	0%	100%
Total_War_WARHAMMER	0.0039	3/562	0%	100%
DOOM	0.0054	5/562	1%	99%
Rocket_League	0.2434	412/562	73%	27%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0118	23/562	4%	96%
ARK_Survival_Evolved	0.0028	7/562	1%	99%
DARK_SOULS_III, The_Witcher_3	0.0044	6/562	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0033	3/562	0%	100%
Battleborn	0.0033	5/562	1%	99%

Table 2.6: Insurgency	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0125	18/663	3%	97%
METAL_GEAR_SOLID_V	0.0038	2/663	0%	100%
War_for_the_Overworld	0.0112	32/663	5%	95%
Stellaris	0.0048	3/663	0%	100%
Total_War_WARHAMMER	0.0044	7/663	1%	99%
DOOM	0.0060	8/663	1%	99%
Rocket_League	0.0086	13/663	2%	98%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.2944	564/663	85%	15%
ARK_Survival_Evolved	0.0023	2/663	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0045	7/663	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0026	2/663	0%	100%
Battleborn	0.0033	5/663	1%	99%

Table 2.7: GTA_V	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.2441	304/376	80%	20%
METAL_GEAR_SOLID_V	0.0054	3/376	0%	100%
War_for_the_Overworld	0.0103	26/376	7%	93%
Stellaris	0.0054	2/376	0%	100%
Total_War_WARHAMMER	0.0053	6/376	1%	99%
DOOM	0.0058	4/376	1%	99%
Rocket_League	0.0064	5/376	1%	99%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0104	12/376	2%	98%
ARK_Survival_Evolved	0.0045	4/376	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0067	7/376	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0032	1/376	0%	100%
Battleborn	0.0025	2/376	0%	100%

Table 2.8: Stellaris	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0095	9/273	3%	97%
METAL_GEAR_SOLID_V	0.0035	0/273	0%	100%
War_for_the_Overworld	0.0125	23/273	8%	92%
Stellaris	0.2774	222/273	81%	19%
Total_War_WARHAMMER	0.0075	4/273	1%	99%
DOOM	0.0038	2/273	1%	99%
Rocket_League	0.0048	7/273	2%	98%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0057	2/273	1%	99%
ARK_Survival_Evolved	0.0029	1/273	1%	99%
DARK_SOULS_III, The_Witcher_3	0.0033	1/273	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0014	0/273	0%	100%
Battleborn	0.0024	2/273	1%	99%

Table 2.9: DOOM	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0109	13/585	2%	98%
METAL_GEAR_SOLID_V	0.0039	1/585	0%	100%
War_for_the_Overworld	0.0097	27/585	5%	95%
Stellaris	0.0055	5/585	1%	99%
Total_War_WARHAMMER	0.0050	6/585	1%	99%
DOOM	0.2906	498/585	85%	15%
Rocket_League	0.0051	5/585	1%	99%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0135	18/585	3%	97%
ARK_Survival_Evolved	0.0016	1/585	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0038	4/585	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0035	3/585	0%	100%
Battleborn	0.0046	4/585	1%	99%

Table 2.10: Total_War_WARHAMMER	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0160	18/321	6%	94%
METAL_GEAR_SOLID_V	0.0030	3/321	1%	99%
War_for_the_Overworld	0.0124	23/321	7%	93%
Stellaris	0.0078	3/321	1%	99%
Total_War_WARHAMMER	0.2742	260/321	81%	19%
DOOM	0.0031	1/321	0%	100%
Rocket_League	0.0033	1/321	0%	100%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0065	3/321	1%	99%
ARK_Survival_Evolved	0.0051	4/321	1%	99%
DARK_SOULS_III, The_Witcher_3	0.0045	3/321	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0021	2/321	0%	100%
Battleborn	0.0026	0/321	0%	100%

Table 2.11: Warhammer_Vermintide	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0109	40/327	12%	88%
METAL_GEAR_SOLID_V	0.0036	10/327	3%	97%
War_for_the_Overworld	0.0120	80/327	24%	76%
Stellaris	0.0063	20/327	6%	93%
Total_War_WARHAMMER	0.0095	28/327	8%	92%
DOOM	0.0080	26/327	8%	92%
Rocket_League	0.0063	25/327	8%	92%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0133	53/327	16%	84%
ARK_Survival_Evolved	0.0014	6/327	2%	98%
DARK_SOULS_III, The_Witcher_3	0.0052	15/327	4%	96%
METAL_GEAR_RISING_REVENGEANCE	0.0040	10/327	3%	97%
Battleborn	0.0040	14/327	4%	96%

Table 2.12: DARK_SOULS_III	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0144	86/489	17%	83%
METAL_GEAR_SOLID_V	0.0050	17/489	3%	97%
War_for_the_Overworld	0.0097	162/489	33%	67%
Stellaris	0.0049	17/489	3%	97%
Total_War_WARHAMMER	0.0048	21/489	4%	96%
DOOM	0.0045	22/489	4%	96%
Rocket_League	0.0046	23/489	4%	96%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0078	50/489	10%	90%
ARK_Survival_Evolved	0.0035	15/489	3%	97%
DARK_SOULS_III, The_Witcher_3	0.0072	38/489	7%	93%
METAL_GEAR_RISING_REVENGEANCE	0.0063	29/489	6%	94%
Battleborn	0.0025	9/489	1%	99%

Table 2.13: METAL_GEAR_RISING_REVENGEANCE	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0122	12/398	3%	97%
METAL_GEAR_SOLID_V	0.0169	21/398	5%	95%
War_for_the_Overworld	0.0084	29/398	7%	93%
Stellaris	0.0055	6/398	2%	98%
Total_War_WARHAMMER	0.0040	2/398	1%	99%
DOOM	0.0052	4/398	1%	99%
Rocket_League	0.0054	5/398	1%	99%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0069	4/398	1%	99%
ARK_Survival_Evolved	8.8137	0/398	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0048	6/398	2%	98%
METAL_GEAR_RISING_REVENGEANCE	0.2626	308/398	77%	23%
Battleborn	0.0028	1/398	1%	99%

Table 2.14: METAL_GEAR_SOLID_V_PHANTOM_PAIN	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0171	22/493	4%	96%
METAL_GEAR_SOLID_V_PHANTOM_PAIN	0.2225	385/493	78%	22%
War_for_the_Overworld	0.0090	32/493	6%	94%
Stellaris	0.0057	6/493	1%	99%
Total_War_WARHAMMER	0.0045	8/493	1%	99%
DOOM	0.0049	2/493	1%	99%
Rocket_League	0.0061	9/493	1%	99%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0112	11/493	2%	98%
ARK_Survival_Evolved	0.0013	0/493	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0067	7/493	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0054	6/493	1%	99%
Battleborn	0.0036	5/493	1%	99%

Table 2.15: Counter_Strike	P(B)	Classified	Hit Rate	Miss Rate
GTA_V	0.0182	59/428	13%	87%
METAL_GEAR_SOLID_V	0.0044	10/428	2%	98%
War_for_the_Overworld	0.0094	120/428	28%	72%
Stellaris	0.0063	11/428	2%	98%
Total_War_WARHAMMER	0.0042	10/428	2%	98%
DOOM	0.0051	15/428	2%	98%
Rocket_League	0.0107	45/428	10%	90%
Insurgency, Warhammer_Vermintide, Counter_Strike	0.0429	123/428	28%	72%
ARK_Survival_Evolved	0.0025	7/428	2%	98%
DARK_SOULS_III, The_Witcher_3	0.0045	14/428	2%	98%
METAL_GEAR_RISING_REVENGEANCE	0.0023	8/428	2%	98%
Battleborn	0.0032	6/428	2%	98%

D Tables 2: Accuracies for every game in partition 3.

Table 3.1: War_for_the_Overworld	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0104	2/93	2%	98%
DOOM, Warhammer_Vermintide	0.0077	0/93	0%	100%
METAL_GEAR_SOLID_V	0.0028	0/93	0%	100%
Stellaris	0.0063	1/93	1%	99%
Total_War_WARHAMMER	0.0049	0/93	0%	100%
War_for_the_Overworld	0.2798	85/93	91%	9%
Rocket_League	0.0039	0/93	0%	100%
ARK_Survival_Evolved	0.0073	2/93	2%	98%
Insurgency, Counter_Strike	0.0058	1/93	1%	99%
DARK_SOULS_III, The_Witcher_3_Wild_Hunt	0.0012	0/93	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0018	0/93	0%	100%
Battleborn	0.0083	2/93	2%	98%

Table 3.2: Battleborn	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0121	16/227	7%	93%
DOOM, Warhammer_Vermintide	0.0061	2/227	1%	99%
METAL_GEAR_SOLID_V	0.0035	2/227	1%	99%
Stellaris	0.0060	0/227	0%	100%
Total_War_WARHAMMER	0.0063	2/227	1%	99%
War_for_the_Overworld	0.0075	5/227	2%	98%
Rocket_League	0.0070	4/227	2%	98%
ARK_Survival_Evolved	0.0011	0/227	0%	100%
Insurgency, Counter_Strike	0.0099	3/227	2%	98%
DARK_SOULS_III, The_Witcher_3	0.0010	0/227	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0032	1/227	1%	99%
Battleborn	0.2899	192/227	84%	16%

Table 3.3: The_Witcher_3	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0142	107/644	16%	84%
DOOM, Warhammer_Vermintide	0.0060	11/644	2%	98%
METAL_GEAR_SOLID_V	0.0047	8/644	1%	99%
Stellaris	0.0071	14/644	2%	98%
Total_War_WARHAMMER	0.0071	14/644	2%	98%
War_for_the_Overworld	0.0080	15/644	2%	98%
Rocket_League	0.0047	12/644	2%	98%
ARK_Survival_Evolved	0.0037	8/644	1%	99%
Insurgency, Counter_Strike	0.0064	17/644	2%	98%
DARK_SOULS_III, The_Witcher_3	0.1799	422/644	65%	35%
METAL_GEAR_RISING_REVENGEANCE	0.0052	11/644	2%	98%
Battleborn	0.0048	5/644	1%	99%

Table 3.4: ARK_Survival_Evolved	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0131	44/523	8%	92%
DOOM, Warhammer_Vermintide	0.0054	9/523	2%	98%
METAL_GEAR_SOLID_V	0.0027	4/523	1%	99%
Stellaris	0.0076	5/523	1%	99%
Total_War_WARHAMMER	0.0065	9/523	2%	98%
War_for_the_Overworld	0.0101	20/523	4%	96%
Rocket_League	0.0061	14/523	3%	97%
ARK_Survival_Evolved	0.2743	404/523	77%	23%
Insurgency, Counter_Strike	0.0056	7/523	2%	98%
DARK_SOULS_III, The_Witcher_3	0.0016	0/523	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0027	1/523	1%	99%
Battleborn	0.0048	6/523	1%	99%

Table 3.5: Rocket_League	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0163	73/562	13%	87%
DOOM, Warhammer_Vermintide	0.0071	8/562	1%	99%
METAL_GEAR_SOLID_V	0.0031	5/562	1%	99%
Stellaris	0.0080	10/562	1%	99%
Total_War_WARHAMMER	0.0051	4/562	1%	99%
War_for_the_Overworld	0.0079	8/562	1%	99%
Rocket_League	0.2268	411/562	73%	27%
ARK_Survival_Evolved	0.0025	2/562	1%	99%
Insurgency, Counter_Strike	0.0093	12/562	2%	98%
DARK_SOULS_III, The_Witcher_3	0.0018	3/562	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0038	4/562	1%	99%
Battleborn	0.0076	22/562	2%	98%

Table 3.6: Insurgency	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0115	35/663	5%	95%
DOOM, Warhammer_Vermintide	0.0101	15/663	2%	98%
METAL_GEAR_SOLID_V	0.0034	5/663	1%	99%
Stellaris	0.0063	2/663	1%	99%
Total_War_WARHAMMER	0.0058	9/663	1%	99%
War_for_the_Overworld	0.0100	9/663	1%	99%
Rocket_League	0.0075	12/663	2%	98%
ARK_Survival_Evolved	0.0027	4/663	1%	99%
Insurgency, Counter_Strike	0.2979	554/663	83%	17%
DARK_SOULS_III, The_Witcher_3	0.0018	3/663	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0028	2/663	1%	99%
Battleborn	0.0083	13/663	2%	98%

Table 3.7: Grand_Theft_Auto_V	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.2485	323/376	86%	14%
DOOM, Warhammer_Vermintide	0.0109	8/376	2%	98%
METAL_GEAR_SOLID_V	0.0058	5/376	1%	99%
Stellaris	0.0070	5/376	1%	99%
Total_War_WARHAMMER	0.0074	5/376	1%	99%
War_for_the_Overworld	0.0091	7/376	2%	98%
Rocket_League	0.0092	8/376	2%	98%
ARK_Survival_Evolved	0.0049	3/376	1%	99%
Insurgency, Counter_Strike	0.0092	6/376	2%	98%
DARK_SOULS_III, The_Witcher_3	0.0031	2/376	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0047	3/376	1%	99%
Battleborn	0.0056	1/376	1%	99%

Table 3.8: Stellaris	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0097	18/273	7%	93%
DOOM, Warhammer_Vermintide	0.0056	2/273	0%	100%
METAL_GEAR_SOLID_V	0.0027	0/273	0%	100%
Stellaris	0.2764	237/273	86%	14%
Total_War_WARHAMMER	0.0088	5/273	2%	98%
War_for_the_Overworld	0.0100	7/273	2%	98%
Rocket_League	0.0052	2/273	0%	100%
ARK_Survival_Evolved	0.0015	0/273	0%	100%
Insurgency, Counter_Strike	0.0059	0/273	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0017	0/273	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0026	0/273	0%	100%
Battleborn	0.0044	2/273	0%	100%

Table 3.9: DOOM	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0110	28/585	5%	95%
DOOM, Warhammer_Vermintide	0.3047	506/585	87%	13%
METAL_GEAR_SOLID_V	0.0027	0/585	0%	100%
Stellaris	0.0056	6/585	1%	99%
Total_War_WARHAMMER	0.0058	3/585	1%	99%
War_for_the_Overworld	0.0092	11/585	2%	98%
Rocket_League	0.0044	4/585	1%	99%
ARK_Survival_Evolved	0.0021	1/585	0%	100%
Insurgency, Counter_Strike	0.0103	12/585	2%	98%
DARK_SOULS_III, The_Witcher_3	0.0021	3/585	1%	99%
METAL_GEAR_RISING_REVENGEANCE	0.0042	4/585	1%	99%
Battleborn	0.0058	7/585	1%	99%

Table 3.10: Total_War_WARHAMMER	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0118	20/321	6%	94%
DOOM, Warhammer_Vermintide	0.0074	3/321	1%	99%
METAL_GEAR_SOLID_V	0.0021	2/321	1%	99%
Stellaris	0.0097	5/321	1%	99%
Total_War_WARHAMMER	0.2897	274/321	85%	15%
War_for_the_Overworld	0.0083	8/321	2%	98%
Rocket_League	0.0055	3/321	1%	99%
ARK_Survival_Evolved	0.0018	3/321	1%	99%
Insurgency, Counter_Strike	0.0052	1/321	1%	99%
DARK_SOULS_III, The_Witcher_3	0.0014	0/321	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.0027	0/321	0%	100%
Battleborn	0.0041	2/321	1%	99%

Table 3.11: Warhammer_Vermintide	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0129	100/327	30%	70%
DOOM, Warhammer_Vermintide	0.0073	27/327	8%	92%
METAL_GEAR_SOLID_V	0.0027	10/327	3%	97%
Stellaris	0.0062	13/327	4%	96%
Total_War_WARHAMMER	0.0119	34/327	10%	90%
War_for_the_Overworld	0.0099	34/327	10%	90%
Rocket_League	0.0058	18/327	5%	95%
ARK_Survival_Evolved	0.0022	7/327	2%	98%
Insurgency, Counter_Strike	0.0082	41/327	13%	97%
DARK_SOULS_III, The_Witcher_3	0.0035	9/327	3%	97%
METAL_GEAR_RISING_REVENGEANCE	0.0031	7/327	2%	98%
Battleborn	0.0068	27/327	8%	92%

Table 3.12: DARK_SOULS_III	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0117	193/489	39%	61%
DOOM, Warhammer_Vermintide	0.0059	36/489	7%	93%
METAL_GEAR_SOLID_V	0.0025	11/489	2%	98%
Stellaris	0.0063	31/489	6%	94%
Total_War_WARHAMMER	0.0071	36/489	7%	93%
War_for_the_Overworld	0.0086	50/489	10%	90%
Rocket_League	0.0044	25/489	5%	95%
ARK_Survival_Evolved	0.0025	9/489	2%	98%
Insurgency, Counter_Strike	0.0053	27/489	5%	95%
DARK_SOULS_III, The_Witcher_3	0.0041	16/489	3%	97%
METAL_GEAR_RISING_REVENGEANCE	0.0074	34/489	8%	92%
Battleborn	0.0047	21/489	4%	96%

Table 3.13: METAL_GEAR_RISING_REVENGEANCE	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0105	35/398	9%	91%
DOOM, Warhammer_Vermintide	0.0082	7/398	2%	98%
METAL_GEAR_SOLID_V	0.0143	19/398	5%	95%
Stellaris	0.0057	4/398	1%	99%
Total_War_WARHAMMER	0.0048	1/398	0%	100%
War_for_the_Overworld	0.0062	2/398	1%	99%
Rocket_League	0.0046	4/398	1%	99%
ARK_Survival_Evolved	0.0019	1/398	0%	100%
Insurgency, Counter_Strike	0.0050	1/398	0%	100%
DARK_SOULS_III, The_Witcher_3	0.0026	1/398	0%	100%
METAL_GEAR_RISING_REVENGEANCE	0.2828	321/398	80%	20%
Battleborn	0.0040	2/398	1%	99%

Table 3.14: METAL_GEAR_SOLID_V	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0175	65/493	13%	87%
DOOM, Warhammer_Vermintide	0.0070	10/493	2%	98%
METAL_GEAR_SOLID_V	0.2279	356/493	72%	28%
Stellaris	0.0078	6/493	1%	99%
Total_War_WARHAMMER	0.0061	3/493	1%	99%
War_for_the_Overworld	0.0075	10/493	2%	98%
Rocket_League	0.0050	4/493	1%	99%
ARK_Survival_Evolved	0.0030	5/493	1%	99%
Insurgency, Counter_Strike	0.0074	4/493	1%	99%
DARK_SOULS_III, The_Witcher_3	0.0037	10/493	2%	98%
METAL_GEAR_RISING_REVENGEANCE	0.0087	17/493	3%	97%
Battleborn	0.0052	3/493	1%	99%

Table 3.15: Counter_Strike	P(B)	Classified	Hit Rate	Miss Rate
Grand_Theft_Auto_V	0.0122	151/428	35%	65%
DOOM, Warhammer_Vermintide	0.0097	24/428	5%	95%
METAL_GEAR_SOLID_V	0.0031	9/428	2%	98%
Stellaris	0.0075	15/428	3%	97%
Total_War_WARHAMMER	0.0059	14/428	3%	97%
War_for_the_Overworld	0.0084	24/428	5%	95%
Rocket_League	0.0112	46/428	10%	90%
ARK_Survival_Evolved	0.0016	8/428	2%	98%
Insurgency, Counter_Strike	0.0372	105/428	24%	76%
DARK_SOULS_III, The_Witcher_3	0.0016	7/428	2%	98%
METAL_GEAR_RISING_REVENGEANCE	0.0034	5/428	1%	99%
Battleborn	0.0074	20/428	5%	95%