

A Search Engine for Hours of Fun and Play

Developing a scalable game search engine



Bachelor thesis by Deborah Maaijen

A Search Engine for Hours of Fun and Play

Developing a scalable game search engine

Author: D.C. Maaijen. studentnumber 2082873
Fontys University of Applied Sciences
HBO ICT Software Engineering

Company: GameHouse Europe
Company supervisor: N. Koek

Teacher supervisor and assessor: A. Lak
Assessor: A. Kaspar
Location: Eindhoven, the Netherlands

The prospectus meeting will be held on January 30, 2012
All images in this thesis are copyright GameHouse®
Document version: 1.8

January 5, 2012

Foreword

In the period of September 19, 2011 until February 3, 2012 I have worked on my thesis project at the casual games company GameHouse Europe in Eindhoven. I have integrated a search functionality into their publicly accessible website that presents the games to their customers. Because it is an international company, I decided with my supervisor to conduct the thesis and internal documentation in English. I would like to thank my on-site supervisor Niels Koek for his time and guidance in the project. I would also like to thank my teacher supervisor Albert Lak for his feedback on the materials. During the first year of studies I received a lot of encouragement and help from Robin Leffmann, so a big thanks goes out to him. Lastly I would like to thank the reviewers Andreas Larsson, David Holz, David Weinehall and Vanessa Ezekowitz for their feedback on the thesis.

Deborah Maaijen,
December 18, 2011, Eindhoven

Contents

1	Introduction	6
2	Company Profile	7
3	Assignment	9
4	Customer Survey	12
4.1	Generic	12
4.2	Suggestions	14
4.3	Feedback evaluation	15
5	Comparing Search Engines	16
5.1	Libraries and full packages	16
5.2	SAAS	18
6	Database and Programming Structure	20
6.1	Programming Structure	20
6.2	The Database	20
7	Search Engine Implementation	23
7.1	Lucene algorithms	25
7.2	Indexing	25
7.3	Searching	26
7.4	Filtering and Sorting	26
7.5	Spell checking	27
7.6	Tags	27
8	Functional Testing	29
8.1	Indexing and search speed	30
9	Conclusion	32
A	Zylom Zoekmachine Survey NL	II
A.1	Survey	II
A.1.1	Voorkeuren	II
A.1.2	Nieuwe zoekoptie	VI

A.1.3	Achtergrond	VI
B	Search Engine comparison	VIII
B.1	Libraries	VIII
B.1.1	The old in-house developed code	VIII
B.1.2	Lucene	IX
B.1.3	Elasticsearch	IX
B.1.4	Solr	X
B.1.5	Hadoop Search, HSearch	XI
B.1.6	Xapian	XI
B.1.7	Sphinx	XII
B.1.8	Regain	XIII
B.2	Search engine services(Software As A Service)	XIII
B.2.1	Google site search	XIII
B.2.2	Yahoo Build your Own Search Service	XIV
C	Documentation	XV
C.1	Lucene Search package	XV
C.1.1	Document Fields	XV
C.1.2	Managing the Index	XVI
C.1.3	Filtering and Sorting	XVI
C.1.4	Searching	XVI
C.2	Lucene Query Syntax	XVI
C.2.1	Enabled by default	XVII
C.3	Extra libraries	XVIII
C.4	Search Term Rating methods	XVIII
C.5	Lucene Lingua Operandi	XVIII
C.6	Tags	XIX
C.7	Problems to be anticipated	XIX
C.7.1	Duplicate entries	XIX
C.7.2	Updating existing documents	XIX
C.7.3	Multiple entries per field	XX
C.7.4	Irrelevant game results	XX
C.7.5	Discrepancy in the documentation	XX

Abstract

The casual games company GameHouse desired to have a fast search engine implemented on their website www.zylom.com. GameHouse is a company located in Eindhoven with around sixty-five employees ranging from game designers to marketers. They have an international website translated in eight languages serving over seven hundred games as their main product. Because there are over five million visitors per month, they needed a fast product implemented, handling multiple search operations at once. Their software environment was written in the programming language Java with styling and other web functionality done in other languages and scripts. They make use of a large Microsoft database to store all of the data. By means of software relying on this database, statistical reports are generated. Several servers handle web traffic, sharing the load and enabling faster access. The search engine should also make use of several servers to divide the load. Other demands were amongst others a spell checking functionality and filtering by game genre, release date and game type. An earlier attempt was made at a search engine and the request was to see if it was usable. Any third party search engine component should preferably be free.

After having been presented with the initial requirements, additional questions arose and they were answered and integrated in the Project Initiation Document. Beside the company's requirements, there were no requirements known from the players of the games. The first logical step was setting up a survey. It was available on the website for 3 days. The survey did not yield results related to specific search functionality, only general remarks.

Research has been done on existing search engines to get familiarized with the concept. Several popular complete search engines and separate modules that had to be integrated have been compared on functionality and the most feasible have been tested. Based on the comparisons, personal experience with the implementation and likely acceptance among the other programmers after a discussion, the standalone programming module Lucene was chosen and integrated into the existing infrastructure. This included separate plug-ins for the extra required functionality. The functionality of the end product at the time of writing included indexing of a dynamic list of games, filtering and sorting games, spell checking functionality and a function to add tags to games in the index. The tags will contain synonyms of game names and relevant words that are not in the descriptions, for more matching possibilities. The functions were tested for speed and accuracy of the results and the statistics were presented.

Almost all of the requirements were met. Searching games by means of a question form, was dropped due to time restrictions. An option to analyze which game players click in the search results might still be implemented after this thesis was finished.

Chapter 1

Introduction

More than 5 million visitors per month turn to www.zylom.com to play games. Over the years the amount of games available on the site has increased, but the amount of players has slightly decreased. Game studio and publisher GameHouse would like to attract and retain more players on the website and adding a search engine is part of this plan. Ease-of-use is more important than offering complex queries as the players are generally not technically savvy. It also has to be fast, being able to handle large volumes of queries and preferably open source so that the IT department can expand the source code in the future. The project also requires the rewriting of the existing presentation class of games in Java.

The first chapters cover information about GameHouse and the project details for this thesis. This is followed by a survey conducted to discover the features that the players desire in chapter 4. Chapter 5 concerns the testing of several search engine packages, libraries and services for suitability to be implemented on the website. The existing database is, due to its vastness, only partially presented in chapter 6. It also contains a basic introduction to the existing software infrastructure. The implementation in chapter 7 describes the new classes, the changed classes and their functionality. Chapter 8 shows the speed and accuracy test results of the product. The conclusion is presented in chapter 9 followed by the appendices: Search Engine comparison, customer survey and the documentation posted on the intranet. The documentation is originally made in Wiki markup language, but is adjusted to the LaTeX typesetting system. The Project Initiation Document (PID) is added as a separate document to keep the front page and glossary separate from the thesis.

Chapter 2

Company Profile

GameHouse Europe
Emmasingel 21-23
5611 AZ Eindhoven
www.zylom.com

Founded in 1998, GameHouse started as a small game studio in Seattle, Washington, the epicenter of casual games, and has expanded over the years with major offices in the Netherlands, UK, France, Germany and Brazil. Today more than 50 million players around the world enjoy these games. The games are offered by means of publishing, licensing, distribution, and retail business. With the largest catalog of 3,000 casual games and counting, they offer many games and more ways to play them - all across the expansive mix of channels including online, download, smartphone, tablets, Facebook, and Google+. The aim is to continually provide new social game experiences that connect players with their friends and families, all on the devices they choose to use. The game portal Zylom, available in 8 languages, has been awarded the *website of the year* award in several countries during several years. Truncated games can be played online, full versions can be purchased individually, or a subscription named FunPass can be bought to play all games for a set duration of time. One of the in-house developed game series is named "Delicious", which is a casual restaurant simulation game. The familiar icon of the games division and website Zylom, is the green and orange clad woman Emily, depicted as a child on the cover page. The game "Delicious - Emily's Childhood memories" won the *Best Casual Game* award in the 2011 Dutch Game Awards. GameHouse is the new name for the Games division Zylom. The website continues to be named Zylom.com. The local organization of the global company consists of about 65 employees and is informal, making communication with all local departments accessible. The main customer demographics are women of age 25 and older.

Staff

The supervisor assigned is Team Lead System Development: Niels Koek, nkoek@real.com. He is responsible for helping with technical questions, being introduced to the company, forming the project initiation document and approving the thesis. He is the manager of the System Development department, responsible for programming the website and maintaining the system.

infrastructure among others. This is where the project is done.

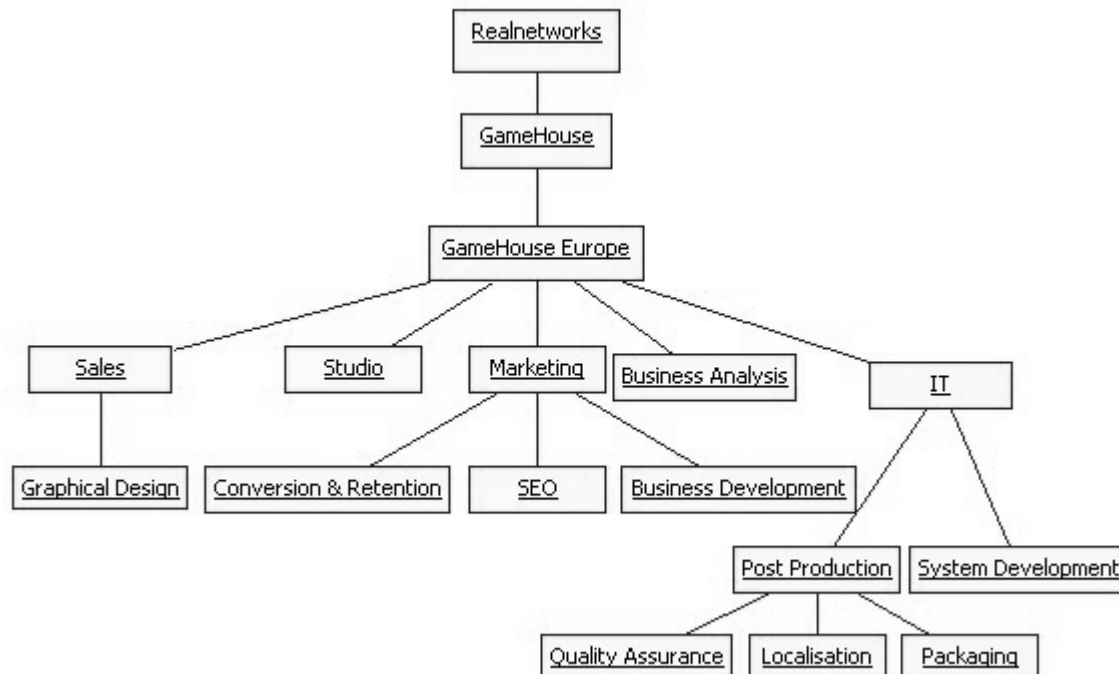


Figure 2.1: The hierarchical structure of the company. The GameHouse organization runs across several countries but almost all support and maintenance personnel work in Eindhoven. The project will mostly cover the IT department, discussing the physical and network environment as well as the website.

Other employee contacts that are relevant to this project are also locally present. Tjerk is the web designer working for the Graphical Design team, who is the contact for the design of the site. From Conversion & Retention, Irma is the contact for gathering the commercial statistical demands. She is also in charge of deciding which features should be added to the site. Anouk is the Search Engine Optimizing specialist, making sure the website will get more visitors and Janne is the copywriter responsible for the translations and other official texts. The Conversion & Retention department is responsible for gaining paying customers and keeping the services attractive for existing customers. It consists of 4 people that have marketing experience, and a graphical designer. The web developers are several team members from the Development team, direct colleagues during this project. The organization is represented in figure 2.1

Chapter 3

Assignment

Zylom.com is the portal by the company GameHouse for hundreds of casual games. The amount of unique visitors to the site seems to be slightly declining over the most recent year.

To attract more visitors, a better search engine position is desirable. Conversion & Retention employee Anouk helps out by taking care Search Engine Optimization and implementing a search engine is a second method.. A search engine is needed in order to make it easier for customers to find a certain game or a certain type of game they are interested in. Currently

players cannot search directly. They make use of a menu structure to browse categories. By means of a search engine, players need less time browsing and therefore can spend more time playing and downloading games, resulting in more sales. The search engine should be able to provide reports as to which search patterns are popular. The company can then use the statistics to create more revenue by meeting the demands. Since most of the other game sites have a search engine, GameHouse needs one to offer the best user experience. There was an earlier attempt at a search engine, but that was aborted. They wanted to know if it is usable in the project. A list of search functionality that they would like to see implemented was given, including the



Figure 3.1: Search design given for the final product.

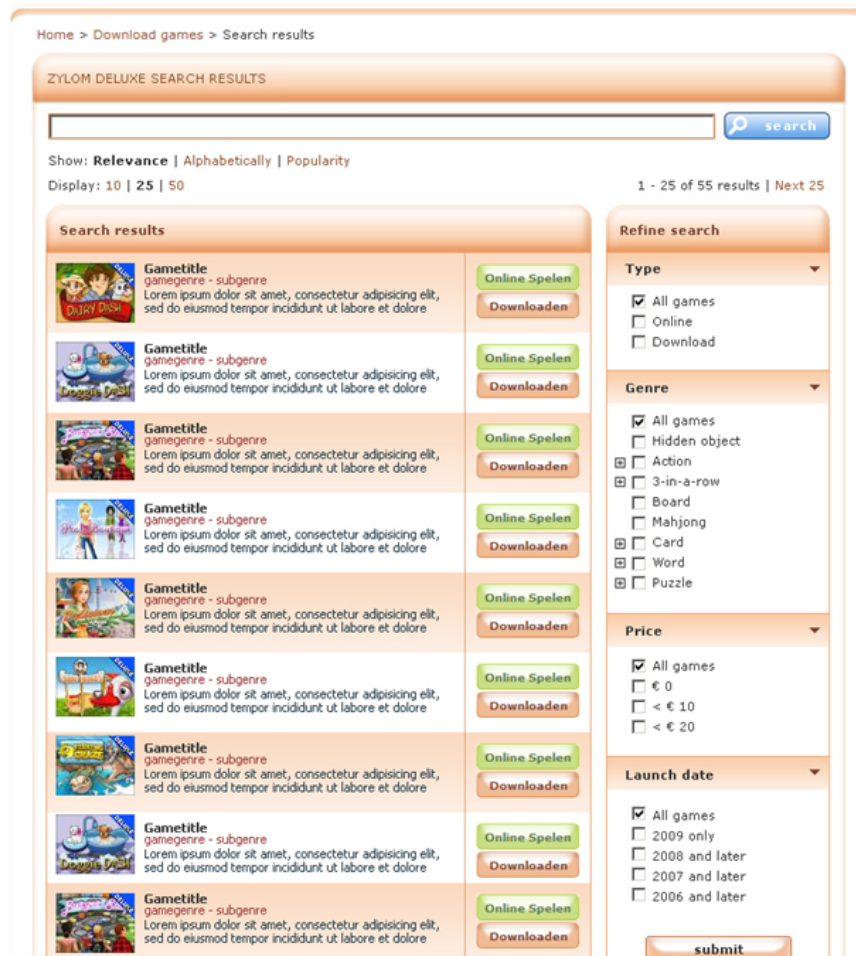


Figure 3.2: Results page and filter design given for the final product.

images in 3.2 and 3.1. Implementing the graphical interface is not part of the task, but is good to see which functionality and which formatting has to be provided. The needed filtering options were eventually decided to be release date ranges of a year, all categories and game type. The categories given to games are puzzle, card, word, action, 3-in-a-row, hidden object, and time management. Hidden object games are point and click games, where a player searches for items hidden between others on her screen. Time management games are simulation games, like the popular restaurant management series 'Delicious'. The rest of the genres are straightforward. Based on the requirements on the following page the project initiation document was designed. Some of the requirements like 'natural language processing' have been dropped after a discussion. It would cost a lot of time to analyze sentences to come up with a game as an answer and it seems more suitable for a FAQ search feature. The project initiation document was added as a separate document.

The demands from GameHouse in short:

- Each country should show a limited number of games, as not every game should be available in every language.
- Natural language processing, so that users can write queries like "I am looking for X". (Requirement dropped).
- Fast performance, as there are thousands of visitors per day.
- It should search intelligently for the most important word in a query string, thus be typo- and word-order tolerant.
- It should be able to handle Unicode text. International characters should be displayed correctly and queries in other languages should be possible.
- It should learn from the queries and store them, so that popular searches can be displayed and auto-corrected.
- Word stemming can provide the feature of having users search for words, and getting the pluralistic forms as a result: "big" can find "bigger" and "biggest" too.
- Synonyms should also give hits. Manual adding of these synonyms should be possible.
- The result pages should be able to achieve high ranking for large, third-party search engine crawler usage, so that the page will attract more visitors.
- Search filters: popularity sort (top sold games for downloads, top clicked for online games), alphabetic sort, filter on release date, genre, online or downloadable games. Optional: search title only, search title and description.
- It would be a merit if the search could have the option to result in an Extensible Markup Language (XML) feed.
- Should be able to boost games manually, so they appear higher in the result list.
- Programmed in Java, because the rest of the system works in Java. (Linux server)
- Database type: Microsoft SQL server for the input and output, because the rest of the data is also stored there. Also because the statistics need to be shown in Microsoft Cubes, which depends on MSSQL.
- The search feature should be able to handle a large volume queries without creating performance issues on the platform. The amount of visitors is over 5 million a month.
- Preferably modular, so that some unneeded resource intensive features can be turned off

Chapter 4

Customer Survey

From Friday October 14 until Monday October 17 an anonymous survey (appendix A) was available on the website for players to fill in. This survey was created in a period of approximately a week. 10 percent of all customers were presented this questionnaire in a pop-up during this time. The number of completed responses were 810 and the uncompleted surveys were 1159 at the time of ceasing the information gathering. The survey was created in a web application that also generated statistics automatically. All of the planned options were presented, so that players would not suggest them again and to see which they would appreciate most. Familiarity with the possibilities of their browsers was inquired after, to find out if players were circumventing the features a search engine would offer. In short, the aim of the survey was to see what players wanted and what their behavior indicates they would need.

4.1 Generic

86% of the respondents were female, which makes 14% male. The majority were aged between 41 and 60 years old (55%), followed by those older than 60 (27%). The majority visited the site several times a day (48%) where the second group biggest group visited a few times per week (37%). 67% never bought a game or had a subscription and 25% bought at least one game (4.1).

Games Only 3.5% did not take into account which game categories they mostly play on Zylom, and 2% didn't mind the subcategories. The overwhelming majority thus picked categories from the list. This is related to a prognosis on the games category filter. It will be used intensively by the majority of customers. The amount of people never browsing on game title, via the alphabetic ordering, is 7%. Regardless of the long list of games, this sorting order looks well used and thus like a good search method for the search engine.

49% of people say that they can find a previous game they played again, followed by 21% who say that sometimes they find the downloadable version instead of the online version or vice versa. Only 3% cannot find the type of game they'd like to play at that moment. 20 people (2%) gave a suggestion, but with the exception of someone complaining about games going missing altogether from the site, they were suggestions that corresponded to the options given. They just saw things like bookmarks as something named differently.

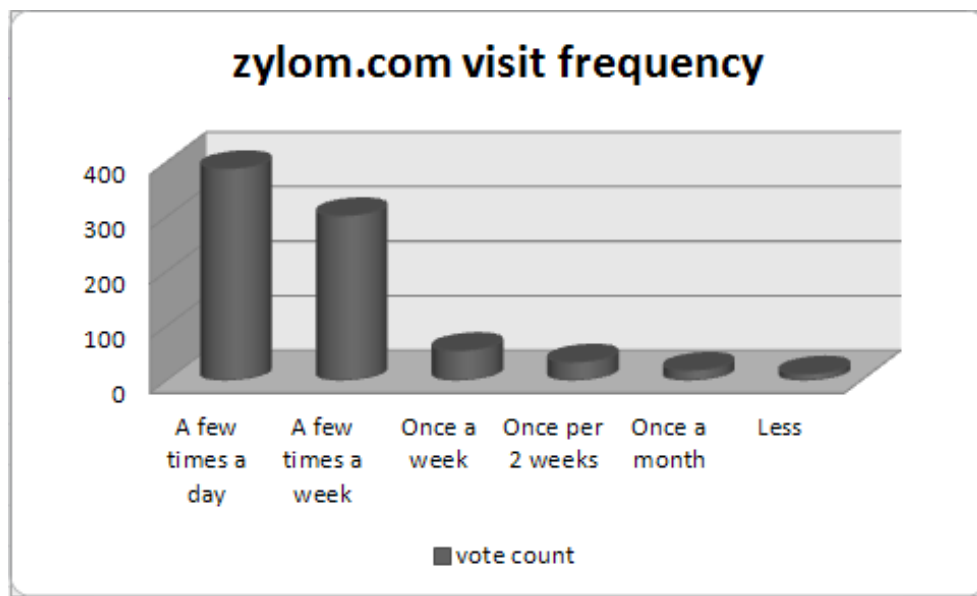


Figure 4.1: The frequency of visits given by players.

Navigational Interestingly, only 20% say they make use of the favorites option of Zylom when logged in. A bigger group of 22% does not know of this possibility. This could be a good item

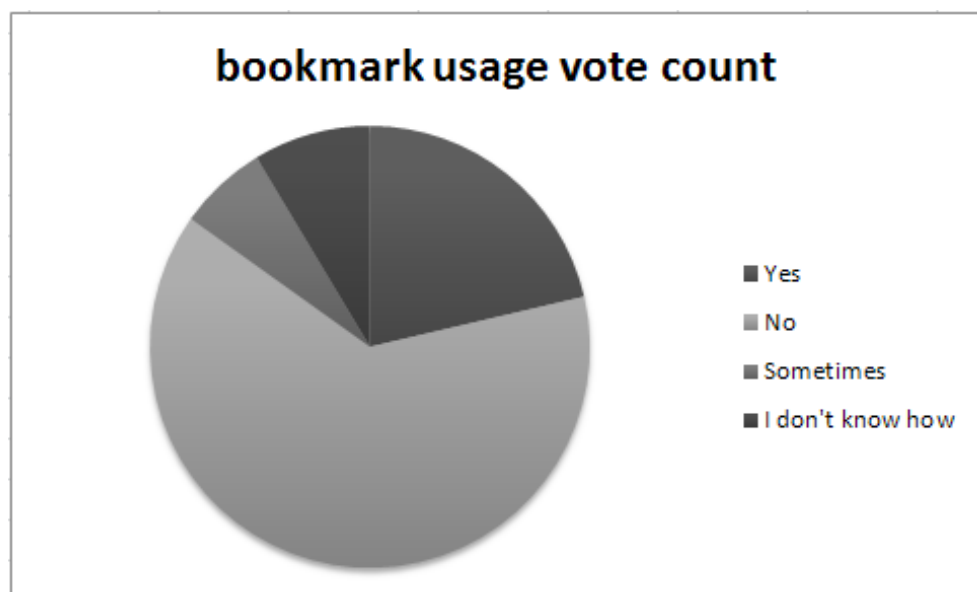


Figure 4.2: Information on bookmark usage given by players.

for the FAQ. This result also comes back in the next section about the suggestions that shows that people don't know how to add games to their favorites list.

Of the respondents, 64% never uses bookmarks of a browser to find games (4.2). 9% indicated interest in learning how to use browser bookmarks for this purpose. 28% already occasionally

make use of bookmarks.

At times 19% of the respondents uses the browser's history to find games, 7% would like to know how to do that. 10.5% frequently use their browser history.

18% make use of their browser's tabs to keep a game page open, to play again. 5% would like to know how to use their tabs.

It takes the majority of respondents between 10 seconds and a minute to find a game they want to play. The remaining 11% take 5 minutes or more to find a game by browsing. 47% say that their search times by browsing are acceptable and the second biggest group (17%) like taking their time to find games. 22% would like a faster browsing experience, 14% don't care about the duration. When given the options of the already planned features for the search engine, only 5% had suggestions. Unfortunately the suggestions of these 55 people did not show up in the survey results page.

4.2 Suggestions

Over the weekend the survey was online, there were many responses that GameHouse could make use of. The vast majority however did not respond to the open questions, which could be a good sign that there are not many complaints on usability of the site. This notion is strengthened by the fact that 31 responses out of 78 on the question "*Is there information you are lacking on the website, that you would like to see added? For example help page and game information?*", say they are content already or have nothing to suggest. A few responses (figure 4.1) were about wanting to see favorites and recent or most played games on the homepage when logging in. Setting favorites is already possible on the profile page, after which they appear on the homepage, so this seems like a candidate for the FAQ. A few of the responses on this question were aimed at the helpdesk, related to technical problems. A lot of the comments however were constructive feedback.

Count	Suggestion
31	Nothing to add or is satisfied with how things are.
9	How to set favorites or see history?
5	How to solve this technical problem?
5	Give information on how long the duration of games is, before they are finished.
3	Want to play the demos for a longer time.
2	Show related games to a certain game.
2	The games don't work sometimes
2	Too many commercials

Table 4.1: Grouped suggestions.

4.3 Feedback evaluation

All of the open questions basically were feedback on issues that were not part of the search engine project, but for general purpose there are some conclusions to make. Based on this survey it is recommended to add information regarding the favorites into the help section and show the favorites and previously played games prominently on the homepage when logged in. The amount of time it approximately takes to finish a game, or a category like "long" "medium" or "short" could be added to a game page, as well as an "add to favorites" option, next to the Facebook "like" button. Also recommended is to have the search engine span the FAQ pages, which is not part of the scope of this project. The outcome of this survey has been given to the Conversion & Retention department, who decides what should go into the search engine and what should change on the website. The full data including the Dutch comments is available as an attachment.

Chapter 5

Comparing Search Engines

This section contains the engines that were referenced the most when searching the web using Google.nl and DuckDuckGo.com starting from September 20, 2011. The search excluding testing went on for about 2 weeks. These references are gathered from recommendations on forums, lists made by others on the web and top results from the search engines. For the full details of the comparison, see the appendix section B.

5.1 Libraries and full packages

Some of the packages are full search engines that will crawl a given website for data and offer a textbox to conduct searches in. Others are libraries that need programming or a separate crawler to obtain and view results in.

The old in-house developed code A JSP page for the Search already existed. It is standardised to use the `doAction()` function with a lot of parameters like the `gamesetID`, `countryID`, `playerID` and the customary `HttpServletRequest` request, `HttpServletResponse` response. `Search.java` extends `GeneralServlet` so that `Search.doAction()` is called by `GeneralServlet.doPost()`, which is in its turn called by `GeneralServlet.doGet()`. The default method called in a servlet is the `doGet()` or `doPost()` function, instead of a constructor. The class responsible for designating tasks to the other search related classes, `SearchManager`, used a cache to store the games. This was updated by a thread and seemed to be rather fast. A `GameComparator`, used for sorting games was also still present. The other classes were `FuzzyString`, `SearchComparator`, `SearchUtils`, `CacheKeyword` and `SearchCache`. This solution used fuzzy word matching to generate hits. The games found by the existing search class did have some correct hits, displaying them neatly in the Java Server Pages (JSP) dedicated to displaying results, but continued with several irrelevant games. The code's sorting functions, a `Comparator` class for games, assumed only one type of game per given comparable list: online or download. However, the search should be for any game, after which the results page shows filter options for game types. The functionality of sorting by release date was completely obsolete.

Lucene Lucene from the Apache Foundation is the package that was eventually chosen to fully integrate in the environment. It was the first one that was clear in its usage and met all

demands. It is a free library that has been active since 2001 and has a large user base, links to implementations shown on their website. It is advertised as high performance and fully written in Java and several of the other popular complete search engine packages were based on this. The external Java package Luke proved to be a helpful tool to test repeated searches in bulk, to view the stored data per document (comparable to a row of data in a database). The progress with integrating the current programming environment with Lucene was fast, so there was no reason to try other libraries.

Solr Solr seemed like a good candidate, praised a lot in the search engine related articles and forums. Solr is based on the Lucene library and also maintained by the Apache Foundation. It advertised its extra features like a comprehensive administrative interface, missing from Lucene, which seems a major asset to getting the other staff to easily manage it. The package came with example data to use with the official tutorial. This worked splendidly. Then an attempt was made to use the crawler Nutch, to see how exactly it will handle the data on the Zylom website. After some effort a post of game data was done via the *curl* command, a command available in Unix-like systems for posting and retrieving data on a network. This was after installing a virtual machine with the operating system CentOS Linux. The full tutorial on the official website was unfortunately outdated, costing unnecessary time to resolve problems via searching the web. The managing tool was very clear and helpful. It offered options to chose several versions of Solr to run, starting and stopping the installation without having to restart an application manager like Tomcat. It shows an overview of defined fields, gives an interface to test queries, show logging and see the Solr XML configuration page. There were a lot of problems trying to connect the database to the program, via the XML configuration options. It continued displaying errors. Defining the fields was difficult because it was very difficult to set the example data apart from the mandatory data in the configuration files. Since it was taking weeks, without any real progress, the attempt to make it work was ceased.

Regain The full search engine package including crawler named Regain is based on the Lucene library. The desktop search and server version were installed for testing. The difference was that in the server version, no graphical user interface was presented to add paths to search in and for changing settings. The installation was easy in both cases and maintenance seemed easy as well. Indexing the site zylom.com/nl was rather fast. However, the most essential thing, the search results, was disappointing. The only available search algorithm type is relevance, which is the matching accuracy. No sorting of the results is implemented. The top results displayed have a relevance rate of up to 168% and they are followed by 15 pages of results that consist of one web page. That page has relevance rate 0% despite that the entered word is actually mentioned and is thus genuinely most relevant. Since the search function itself is so broken, the main thing this is good for is an interface to create another search engine behind.

Elasticsearch In a list of Java search engines, another Lucene based package named Elasticsearch was mentioned.

<http://www.elasticsearch.org> Despite the relatively few companies listed on the site that uses this engine, namely 8, the website looked well maintained enough to investigate the product. It offers tagging, logging, highlighting and outputs its data in the JSON format. A Java Application programming interface (API) was available, so there was a means to integrate it into the back

end. At first, setting up a demo would be a good way to see how this package acts. However, the manual was confusing, even if abundantly documented. The *curl* program was used in every example. At the time, there was no virtual machine with Linux running. As an alternative, using *curl* in the Windows command line via the 'Unix on Windows' program Cygwin was attempted. However the documentation on the homepage was too vague on how exactly to start. Where and which jar files should be extracted where, how to start entering data, how to set up a simple setup etcetera was not mentioned. Instead it jumped in on demonstrating what it does. All of it was done on the command line, like it was meant to be for text browsers. No preview of what was posted in a normal HTML page. There was no explanation of how these JavaScript Object Notation (JSON) formatted indexes were stored to be modified afterwards, and the commands were given in runtime. Nor how normally queries should work, so that for instance filtering out special characters would work. Besides this, the demands were not fully met, as it is a NoSQL schema free program. There was no clue on how to address the database directly. The documentation on the Java API only included defining nodes, to have a distributed environment for the search engine. All in all since it lacked a clear first set-up explanation, especially how to address the database via the Java API nor directly, this package was dropped.

Other search engines There were other search engines that were implemented on websites with high user or data volumes, namely Xapian, HSearch and Sphinx. They seemed eligible as they have already proven their stability and performance on these websites. However they either are built on C or C++ as the programming language or they are designed to be used with another database. Converting data from and to another database generates unnecessary overhead. If the package or library is written in C, even in combination with an API for Java it seems less convenient for the pure Java environment. Considering the aforementioned Java-based packages and libraries as well as the in house expertise, there is no merit there.

5.2 SAAS

These are services that are available through proprietary software, where one pays an amount for the usage of a remotely hosted service, instead of having full unlimited control over the software locally. The database and programming language requirements are thus not relevant in this case. During the course of comparing search engines, several standalone candidates were soon found that already suit the needs, making these expensive services not a cost-efficient option. This is especially the case for the large volume of queries on Zylom.com, as the payment method is based on the amount of times the service is used. Another issue is that the source of the engines is not available to integrate existing Java code on the website with the program. Software as a service (SAAS) is not flexible in that sense.

Google site search Google's service <http://www.google.com/sitesearch/> provides a service for a given site, with cached indexed results. Tags can be used by users to filter results, the scope of the webpages for the search can be defined, newer pages can be prioritized, all 8 languages on Zylom.com are supported and users can search with word synonyms. Other features are labeling the search results and using XML for formatting the results to match the style of the website. Support is available by email, as well as advanced developer documentation and online training. What it does not provide is data on which terms were sought after and how often, and

this is a very important part of the wishes of the sales department. Considering the amount of traffic that comes to the site, the pricing will be at least 2000 Euro annually (price for 500.000 queries). Since this amount would be breached since there are a million visits every month, the specific prices would be higher.

Yahoo BOSS Yahoo! Build your Own Search Service <http://developer.yahoo.com/search/boss/> is another search engine vendor's SAAS solution. Daily usage statistics are shown through the dashboard, results can be re-ordered, the presentation of the results is customizable and user authentication is needed for adjusting the search settings for security reasons. Support is given by dedicated forums. There are two possibilities: 'Boss Hosted Search' and 'Search Boss API'. The first allows configuring it on the Yahoo! site and making it available through javascript on the intended website, thus offering no customizable designing of the results. It is free, but provides its own advertisements between the results. GameHouse already has a business deal with other parties for advertising between game results, so this is not desirable. The API allows the building of search products that uses Yahoo! search technology. The API works with query based pricing for mixed content(text and images) is based per 1000 queries with max 50 results per query. A conservative estimate for a million visitors or queries per month with 200 results or games per query amounts to \$3200 per month.

Chapter 6

Database and Programming Structure

6.1 Programming Structure

The classes in the back end of the system are categorized in neat packages per related object groups, for instance: game (figure 6.1), session, page, data gathering and the new project's packages tag and search. The classes inside these packages are many, there are for instance 29 classes in the game package alone. Since visualizing these packages can easily fill up the whole thesis and describing all contains much irrelevant information, this chapter only shows a small selection of the game package classes mentioned. There are two types of games, downloadable games and online games. The online games are mostly limited versions of the downloadable games, with commercials between levels. The games are obtained from the database and stored in cache by the `DownloadableGameManager` instance, for the objects of type `DownloadableGame` or for the online `Game` objects: the `GameManager`. The `DownloadableGame` and `Game` classes are oriented on the executable side of the games. The classes that handle the representation of the games, with their names, descriptions and help texts displayed are `DownloadableSkinnedGame` and `SkinnedGame`. All games are translated and then assigned a 'game set'. The `GameSet` class can retrieve games separated per language and country. For example: the attribute `gameSetID` with value 2 represents English for the United States. Each game has a category, represented in the `ZylomCategory` class, which is managed by the `ZylomCategoryManager` class. This pattern of (Name of object) Manager continues throughout the code. The managers are static instances to retrieve the objects from a cache. The game developers are represented in their own class: `GameDeveloper`, with again a `GameDeveloperManager`. These are used to programmatically determine which developer created a game or if a developer is still active. Other classes are amongst others related to localization, locations, logging and the help pages.

6.2 The Database

The database will be based on existing tables of a Microsoft SQL database (figure 6.2). Since the website already exists with a database attached, only limited aspects of the plan were modeled. For the search process a diagram has been made in table (7.1). A sequence diagram for the

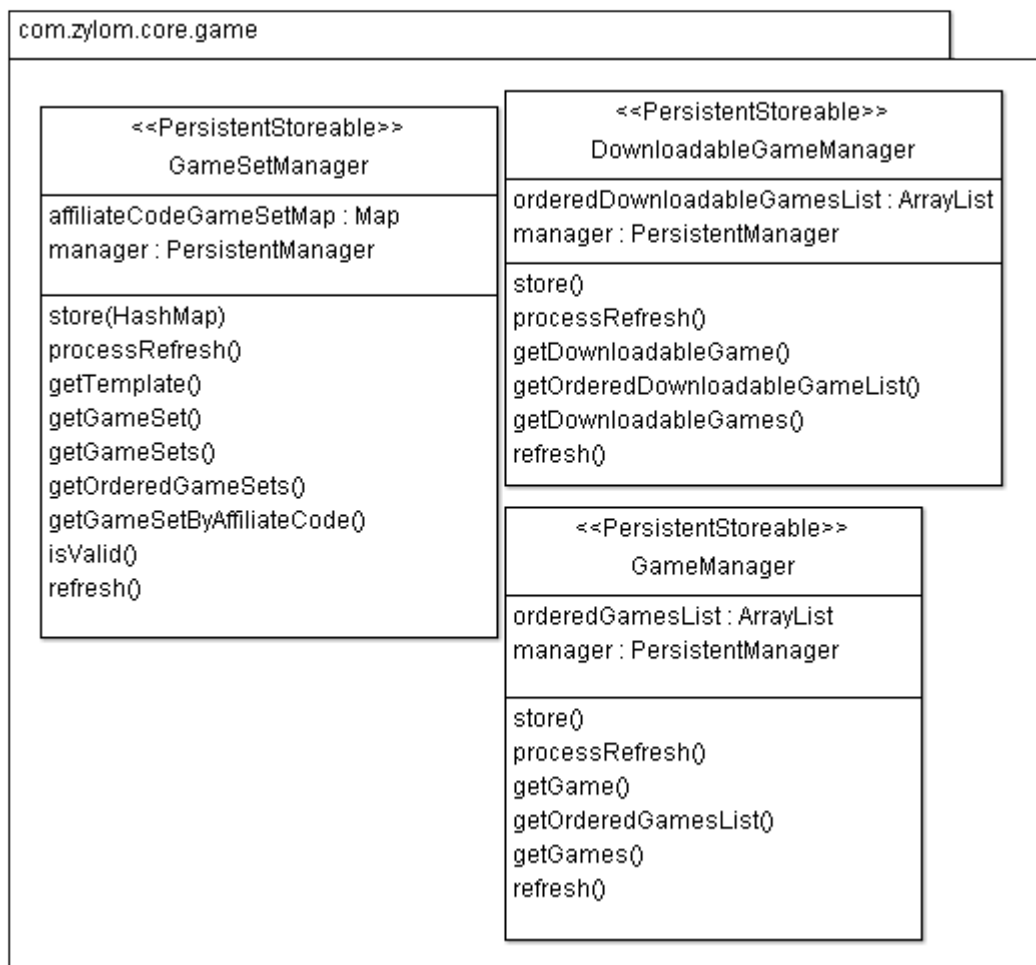


Figure 6.1: Class diagram of some of the games related classes
The partial representation of the game related class structure of the Zylom website.

search process and the filters seemed appropriate, as well as a class diagram representing relations between the tables. The tables TAG and TAGLINK are added for new functionality. TAGLINK consists of *foreign key* attributes: identifying keys from other tables. TAG contains the primary, unique key TAGID, the text like "Dice game" is stored in NAME. The query data is stored in the LOG_SEARCHQUERY table as soon as the user presses enter after entering it in a textfield. The words in the query that do not exist in the index are stored in the TAG table for reviewing. Some of these might be eligible as a tag to games. VISIBLE can be set to TRUE if the tag name is a suitable word for display on the game's page, in contrast to only being used as a match to a game. The LANGUAGEID value is important as the site works with several countries and languages.

Chapter 7

Search Engine Implementation

As mentioned in the chapter 'Comparing Search Engines', the final choice was to use Lucene 3.4.0 as the base of the search engine implementation. The graphical user interface has been

Name	Searching for a game
Summary	A query is entered and a result is given
Actors	Player
Assumptions	Player is looking for a specific game or game.
Description	(1) Player enters text in the search field. (2) Player hits the search button. (3) A list of matching HTML pages of games will be displayed and the query is stored in the database. (3a) If no text is entered, all games will be shown. (3b) Player wants to filter these results further and marks one of the search filters or uses sort mechanisms. (4) User clicks on one of the links in the results.
Exceptions	(2b) Player enters invalid characters; the player will be requested to not use a list of specific characters and try again. (2c) Match is not found, because the game is not available in that language or does not exist. Then either a word is suggested that is similar to the one given in the form of "Did you mean YYY?", or there is no match at all and the search process ends. If there are suggestions, user can click the words to start a new search with those terms
Result	Player gets search results, in the form of click-able links, query is stored in the database.

Table 7.1: Search process use case diagram.

designed after the figures in chapter 3. All the relevant classes are located in the packages `com.zylom.website`.
`search` and `com.zylom.website.search.servlet`, see figure 7.1. The `GameComparator` class is

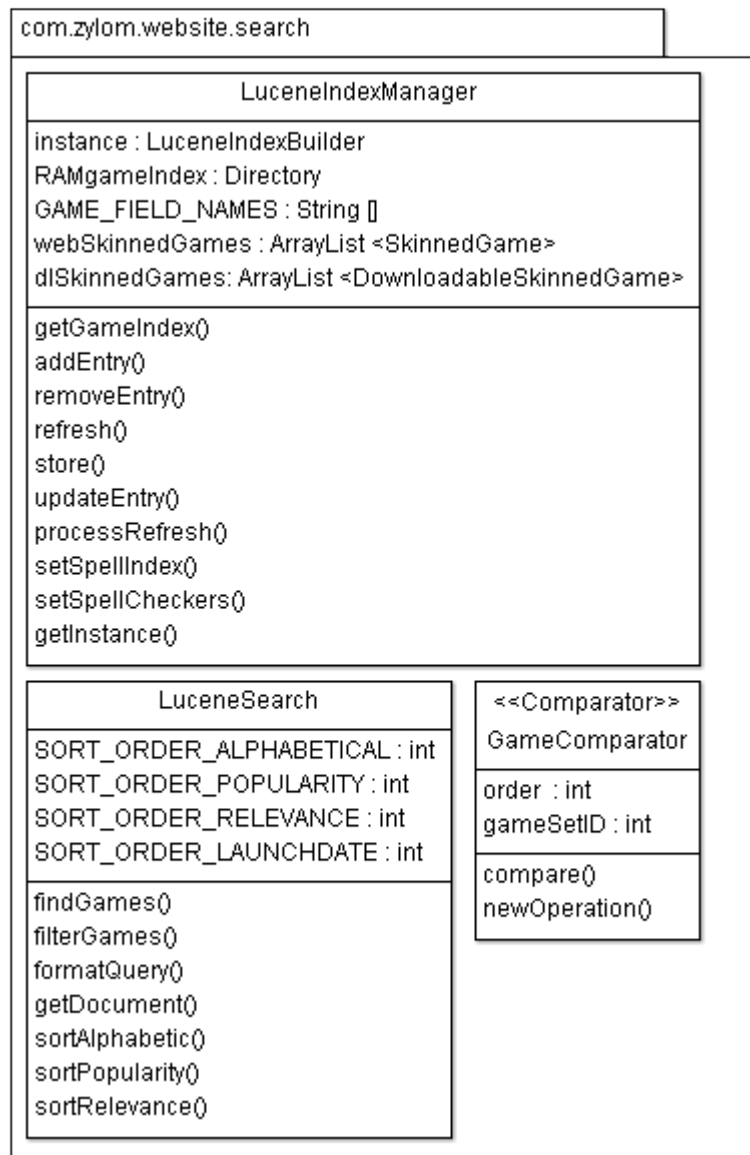


Figure 7.1: Class diagram of the search related classes.

used for comparing both online and downloadable games by name and by release date. The class diagrams in this chapter lack parameters, because there are so many parameters given that it would not fit the pages nicely. Many of the parameters also mean nothing without an explanation and this would be a lot of extra information, whilst the volume of this thesis was already exceeding limit.

7.1 Lucene algorithms

In the official scoring article, the following is said about the algorithms: *"Lucene scoring uses a combination of the Vector Space Model of Information Retrieval and the Boolean model to determine how relevant a given Document is to a User's query. In general, the idea behind the Vector Space Model is the more times a query term appears in a document relative to the number of times the term appears in all the documents in the collection, the more relevant that document is to the query. It uses the Boolean model to first narrow down the documents that need to be scored based on the use of boolean logic in the Query specification. Lucene also adds some capabilities and refinements onto this model to support boolean and fuzzy searching, but it essentially remains a Vector Space Model based system at the heart."* http://lucene.apache.org/java/2_9_1/scoring.html For spell checking functionality the Levenstein, Jaro Wrinkler and N-Gram algorithms were available. The default algorithm was Levenstein. They all measured the distance between a given string and one in the spelling index and gave it a floating point score between 0 and 1. If the result is 1 the string is identical, if it is 0, the distance is maximal. See section C.2.1

7.2 Indexing

The `LuceneIndexManager` class indexes the game data from a cache of currently active, published games. It also created indexes for spell checking functionality, by indexing the game description texts for each individual language to be used with the external `SpellChecker` class. The process of indexing the data needed for parsing a query includes storing the games' title, description, Game Set ID, release date, help text, game genre, the type of game (online or downloadable), and language ID. Before these data are added to the index, they have to fulfill several criteria.

Indexing process

1. The process first limits the amount of game sets that will be iterated, excluding the third party games. Their gameset ID is above 51, or have "Affiliate" in their names.
2. It filters out games that are exclusively promotional. Those games contain the word promo in their name,
3. Checks if they are not obsolete by calling an existing method,
4. Replaces the help text with the name of the game, to boost the words and avoid an empty field in the index
5. Gets and formats the Date object to a normal string of text like 20091231 and strips the dash from the date. This format is obligatory for Lucene's filter function.
6. Any colon in the name of a game is removed
7. The game is added to the index
8. The description of the game is added to a spelling index, stored by language identifier. For example 'enspellchecker' for English.

Adding data to the index is done through the `LuceneManager.addEntry()` method. First a writer class for the index `Directory` object is used. This `IndexWriter` uses an `Analyzer` instance, that is a localized external library for instance `org.apache.lucene.analysis.en.EnglishAnalyzer`. The analyzers tokenize the indexed strings. It also takes care of matching words by recognizing the stem of the words (word stemming), hence every language has its own analyzer. The rows of the index also have the function `setBoost()`, which could be used to boost a row (Lucene's `Document` instance containing information about one game).

7.3 Searching

The class `LuceneSearch` as its name implies, handles all of the searching. It starts in the `Search.java` servlet, where first the `LuceneSearch.formatQuery()` method is called. This removes excess spaces around the query and filters out any special characters except quotes. It also adds 'AND' or 'OR' between words. Then the journey continues to `LuceneSearch.findGames()` that retrieves the data from the index and retrieves the corresponding games, storing them in an `ArrayList`. After storing the query in the database via an existing cached function, the data is sent by `search.java` to `searchresults.jsp`. It is the refurbished version of the JSP that normally displays games. It has a changed layout and it shows both types of games, while the normal version only displays either downloadable games or online games as individual entries. If the first game in the list is an online game, the downloadable version is automatically retrieved from the list of games by game ID, and vice versa. To avoid duplicate entries, the other version of a game gets removed from the array list. So for instance with a list containing games (game1online, game2online, game1download), game1online would group together with game1download. Game2online would group with game2download and as a duplicate game1download would retrieve game1online. This is best illustrated in figure 3.2.

7.4 Filtering and Sorting

Filtering starts in `searchfilter.jsp` which displays the filter options seen in figure 3.2 in chapter 3. Then the data gets posted to the servlet (figure 7.2) `Filter.java` which reads the players chosen options stores these in an `ArrayList`. `Filter.java` includes this `ArrayList` as a parameter `LuceneSearch.filterGames()` for filtering. The filtering happens by translat-

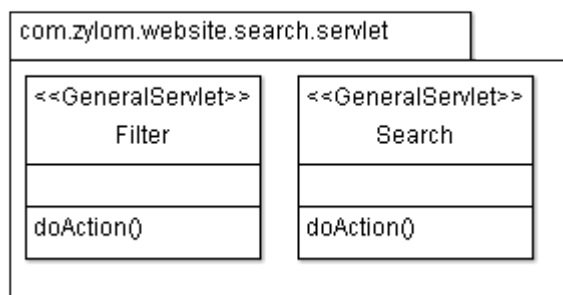


Figure 7.2: Class diagram of the Search Servlet package.

ing every option to something for the search string. For example *gamename:Emily gameset:4* becomes *gamename:Emily gameset:4 launchdate:20101231-20111231 category:4*. Then it calls `LuceneSearch.findGames()` like any other query to get the `ArrayList` of resulting games.

Sorting is done in `LuceneSearch.sort('sorttype')`. In the case of the launch date and alphabetically, a given `ArrayList` is sorted via the `GameComparator` class. The sorting methods on the game displaying page have been recreated, as it used to get **all** games and sort those in the same method. Now the list of matched games are displayed. The old comparator class has been adjusted from its previous state to work with both downloadable and online games. `sortRelevance()` simply does a new query. `sortPopularity()` first divides the games into a Downloadable and Online games list. Then each uses the relevant version of `getGamesList()` and applies `gamesList.retainAll('earlierGameSearchResults')` and concatenates the two sorted games lists into one.

7.5 Spell checking

As mentioned before, the class `SpellChecker` handles spell checking in Lucene. First a separate index per language for all of the descriptions of the games was created. This data was stored in `LuceneIndexManager` while iterating individual game descriptions for the main game index. The individual words were used as a `LucenDictionary` to compare a player-entered string of text with. The `SpellChecker` instances per language were stored in an `ArrayList` by language ID and were retrievable by the `getSpellChecker()` method. Each `SpellChecker` instance could give a corrected word by calling `suggestSimilar(search term, number of results)`. The result was shown as a 'Did you mean' suggestion to the players, when the search generated no results.

7.6 Tags

The new package `com.zylom.website.tag` (figure 7.3) has been added that contains the classes `Tag` and `TagManager`. The format works after example of the other base classes and Managers: implementing the `PersistentStoreable` interfaces, getting data from the database, equivalent getters and setters like `getTag()` and `getTagsbyGameID()`. These tags will be used to give extra keywords to games, that can be used via the search index. They can either be visually displayed on the game page or just indexed. Since games of a certain category can get the same tags, a `getTagsByZylomCategoryID()` method has been added. Descriptive and tag meta data is added to the search page, making direct hits via third party search engines more likely. This should attract more visitors.

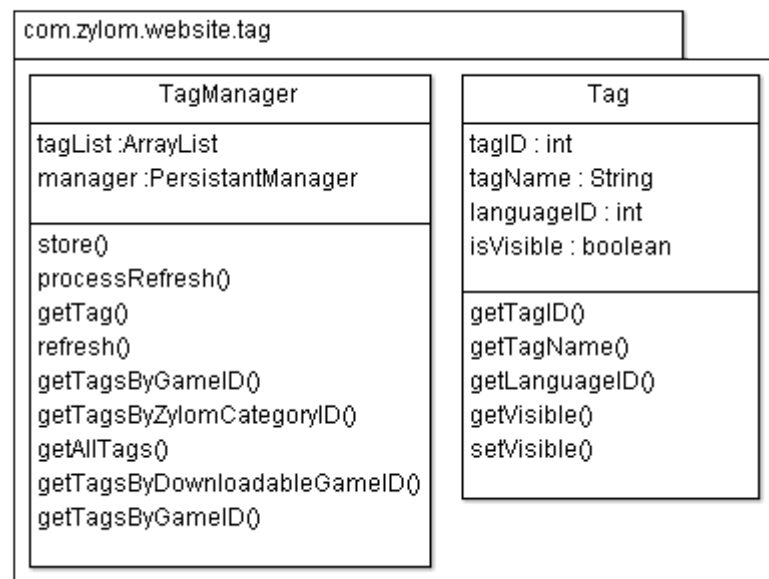


Figure 7.3: Class diagram of the Tag package.

Chapter 8

Functional Testing

The final product was tested on several points for functionality and speed. The test was conducted on a desktop computer running Windows XP Professional with 2 gigabytes of memory and a 2.13 gigahertz processor. The final implementation will be on a virtual server running Ubuntu Linux with 8 gigabytes of memory. Important was that the behavior of the code would work in an environment where there is no use of sessions on a single server, but merely individual requests and response which could be distributed across different servers. It had to process the requests quickly, and be fault tolerant in that operations will continue when exceptions would be encountered. For testing the quality of the results, certain known games have been searched for. If they are all found, it is a positive result. Testing for false positives was done as well.

Logging Each query is logged in the database through an existing class that handles all logs of a player's data. That class is also used to log errors, so that was used for logging events like new languages and their missing specific libraries. The stored data has been checked.

Data integrity The outcome of the search has been tested, by means of taking a few popular games, seeing how many versions there are and then checking if all versions if any show up. Initially only existing terms were used to test the game finding and filtering capabilities. By conducting a more comprehensive test in December, several games seemed to be missing versions or missing altogether. The games' data was more accurately checked before adding it to the index. Other games were found that lacked a description and they were ignored. If it lacked this information it would skip the game, assuming it was obsolete. The problem of incomplete game information was due to using a test database that was incomplete. The searches were also inaccurate if the first word did not occur in the index at all, like 'Dilbert'. By adding boolean statements like AND and OR between words and adding braces around the statements this was solved. For example, Lucene parsed the statement 'gamename:Dilbert farm game ' for a game-title-only search as

gamename:Dilbert AND description:farm OR game

instead of

gamename:(Dilbert OR farm OR game)

Since the description is ignored in the game-title-only search, no results were given. After solving these issues, a second round of tests were done. Searching for the Delicious, Mortimer and

Farm Frenzy series of games, all were found. Searching for games that were only available as a download or online were also correctly displayed. When testing for false positives, none were found. Every hit had at least one of the keywords in the game's name or description. The spell checker used the Levenstein distance as the default string compare algorithm. This was rather inaccurate during the time of writing, suggesting words that were not remotely alike. Since there was a month left to make this work, it was still work in progress. Other algorithms were still an option.

8.1 Indexing and search speed

Another factor of importance is the speed of building the index. Even if building it from scratch isn't done very frequently for updates (at most around 6 times a day), the environment dictates that it has to happen fast. All of the updated data on for instance a new game will be updated sequentially and the other processes should not be significantly slowed down because of the rebuilding of the index. When creating the index on hard disk, the average duration was 38 minutes. This while the index on RAM was an average of 5 minutes. The when stored on hard disk, the size of the game index was 13.5 megabytes and the size of the index with Dutch game descriptions was 1.37 megabytes. A speed test was done on several functions (figure 8.1).

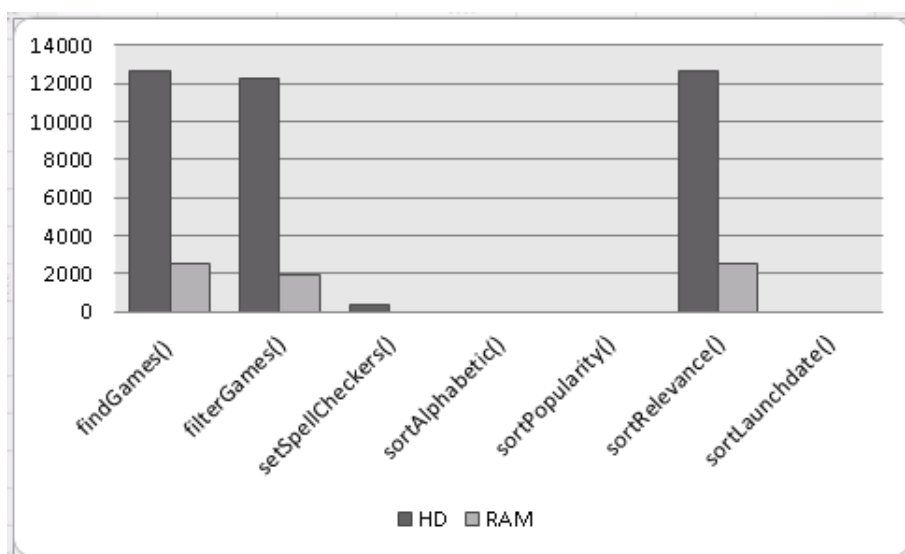


Figure 8.1: Chart displaying the difference in speed of functions between the hard disk and RAM stored indexes.

The results are the average of calling the function 5 times while matching all games. Since the method `sortRelevance()` equals the same actions as `findGames()` when retrieving all games, the duration is the same. The method `setSpellCheckers()` is only done on hard disk since the indexing is conducted separately from linking the index to a `SpellChecker` instance. The `setSpellCheckers()` function looks up the individual language indexes to link each to its own `SpellChecker`. When filtering, the filter chosen was "display all games from 2011 and newer"

resulting in 52 games. The sorting of an existing list of games took 0 milliseconds in both the hard disk and the RAM stored index. A batch operation was done with the third-party tool Luke on the indexes directly. Searching for the Dutch word "en" (Eng:and) in the Dutch spelling index gave 4047 results. Repeating this 200 times gave an average search time of 893 milliseconds. The same batch operation in the game index that contains all languages gives 3434 results and took 1201 milliseconds.

Chapter 9

Conclusion

GameHouse wanted to have a fast, scalable search engine fully integrated into the current Java environment and dynamically updated. This has been achieved by using a separate library and plug-ins as well as reusing old code and creating the rest from scratch. The result of the project were two new JSPs, four new plain Java classes and two new servlets. The JSPs, `searchresults.java` and `searchfilter.java`, present the games in a list on the website and the filter options in a separate box on the side. In the option to filter by release date, the years are dynamically obtained by using system calls. A plain Java class (`LuceneSearch.java`) builds the index and another (`LuceneIndexManager`) handles search functionality. While indexing, each language uses its own analyzer that tokenizes the strings into words. This provides the required word stemming. The index is built using an existing cached list of games to not bring extra load on the database. To optimally use the indexing methods, the spell checking index was built from game descriptions during the games indexing process. The spell checking functionality was required for correcting the user entered queries. A servlet (`Filter.java`) was used to comply to the requirement of filtering games. Another servlet (`Search.java`) handles new search requests. The core of the search engine has become the free Open Source library Lucene since it met all demands of speed, scalability, price (no purchase was needed in the whole project) and functionality. It works together with the existing code on their distributed Tomcat Java application server. From an earlier attempt to write a search engine, only the game comparator class was reused by rewriting it. The company also liked to associate pages and games themselves with extra words. For instance, associating a game with a synonym that wasn't in its description. For handling these tags two Java classes were built. In the company's style of object oriented programming the new classes `Tag` holds the tag information, like the ID and name (synonym word) and a `TagManager` class which retrieves the data from the SQL server database and stores the tags in a cache. It was also important to have the means to resolve any issues that would occur after leaving the project. The functionality of every written Java class and method has documentation that will be included in the generated Javadocs. The documentation on the internal network of GameHouse contains information on the methods used per search task, information on how Lucene handles data and some scenarios on what could go wrong when adjusting the program. To keep the risks of undiscovered bugs at a minimum, the search and extra functionality have been tested for speed and accuracy. The optional feature that did not make it before finishing this report was analyzing the game a player chose when presented a list of results. The required method that was not yet fully tested was `setEntry()` which should boost the value of a given game or add a tag to it.

Evaluation

GameHouse had an interesting environment to work in. The atmosphere had a mix of commercial and creative roles, where my previous working experience was more service oriented. This was the first time I worked as a programmer, previous jobs were in system administration and support. In the first weeks my expectation was that I could write code for the games search functionality and add the help search functionality too. There was not enough time for that however. I greatly underestimated the time to finish, document and debug this. I felt like an appreciated participant in the organization as most of my recommendations about the project's implementation were taken seriously. In the end of the project I wrote the test plan for the final product to measure the speed of the most important functions and to add more research value to the project. Objectively determining what an acceptable speed or duration is for a function was difficult, so the the most important decisive factor lay in approval by the other technical staff. The project was a useful experience that had enough challenges for me to apply creative solutions. Since there was a year between the previous Java classes and the graduation, some of the knowledge was lost, but extra studying solved this. New concepts were learning about Java Server Pages and servlets, as well as getting insights on how search engines work. An extra bonus was that I remained in my own area of expertise by using open source throughout the project. A few times I made the mistake of not being precise and sloppily assuming code would work, but I have seen how much time recovering that mistake took. The search text for example failed when tested with empty spaces before and after the strings. Testing this outside the Netbeans IDEs debugger by displaying the values made it harder to discover the cause. It was also a bad practice to not take breaks from programming to work on the thesis. After a day or two of not looking at the code, I recognized problems that I missed before. My Java knowledge and skills have greatly improved and holding on to thorough testing, documenting and meaningful variable names paid off. At the moment of finishing the thesis there were still three weeks left to program and solve any errors that could occur during the live implementation. The code is expected to go live later in January.

Glossary

Apache Foundation From the official website: the Apache Software Foundation provides organizational, legal, and financial support for a broad range of open source software projects. Through a collaborative and meritocratic development process, Apache projects deliver enterprise-grade, freely available software products that attract large communities of users. The pragmatic Apache License makes it easy for all users, commercial and individual, to deploy Apache products. 15

casual games A genre of games that is noted by its simple controls and a shorter duration to finish than other genres. Mastering these games usually does not require any training or much determination, making them accessible to a general audience with little time. 6, 8

Comparator A Java interface which can be implemented in a class, that can compare classes of the same type by certain attributes. 15

Conversion & Retention The department responsible to keeping customers satisfied and generating more revenue. 7, 8, 14

crawler Also known as spider. Software that automatically gathers text and links from on or more web sites, and stores it for searching purposes. XIII, 10, 15, 16

defunct in Java version 7 Several bugs were included in Java 7 that could crash the Java Virtual Machine in which the applications run. They had to do with executing loops in the code amongst others. See http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=7044738. VIII

Game Set GameHouse's data unit that contains games per language. 19, 24

Jaro Wrinkler Lucene's version of the Jaro Wrinkler distance represents the distance between two strings by boosting the accuracy of the string if the first part of the word matches character by character. It divides the string in a given number of segments and sees how much of a percentage matches. It also does character matching when located elsewhere in the string. 23

Levenstein The Levenstein distance represents the distance between two strings. It is the default method of comparison in the `SpellChecker.suggestSimilar(string of text)` function. It measures the minimum number of changes needed on characters to match one string to another. There is a warning in the original documentation that this might not give the most accurate result, if only one match is retrieved. 23, 29

- Linux** An operating system based on the Linux kernel, consists of GNU and Linux programs. GNU is a project to creating open source software, that runs on Unix-like systems. 16, 17
- N-Gram** The N-Gram distance measures the distance between two strings by means of position based matching of an 'n' number of sequential characters. 23
- Netbeans** A free open source Integrated Development Environment by Sun Microsystems. 32
- Search Engine Optimization** The process of achieving higher ranking in the results of a search engine. 8
- tokenize** Break the string into individual words and remove excess spaces. 25
- Tomcat** Apache's open source Java application server. Hosts servlets and Java server pages. XIII, 16
- Unicode** An international text encoding standard. Text and symbols are generated by means of certain standards of interpreting bytes. Where some of these encoding standards can only display limited characters of western European languages, UTF can display any type of language. 10
- Vector Space Model** *"An algebraic model representing textual information as a vector, the components of this vector could represent the importance of a term (tfidf) or even the absence or presence (Bag of Words) of it in a document"*
<http://pyevolve.sourceforge.net/wordpress/?tag=vector-space-model>. 23
- virtual server** A server that is hosted inside another operating system and is accessible like a normal program. 28

Acronyms

API Application programming interface. 16–18

JSON JavaScript Object Notation. 17

JSP Java Server Pages. 15, 25, 31

PID Project Initiation Document. 5

SAAS Software as a service. 17, 18

XML Extensible Markup Language. 10, 16, 17

Appendix A

Zylom Zoekmachine Survey NL

Pop-up

Beste speler,

We zijn bezig met het vernieuwen van onze website, zodat je je favoriete spelletjes nog beter en sneller kunt vinden. Daarvoor willen we graag weten wat voor spellen je leuk vindt, en hoe je spellen zoekt en vindt op onze website. Wil je een paar vragen voor ons beantwoorden? Dit neemt maar enkele minuten van je tijd in beslag.

Alvast bedankt!

Het Zylom-team

Intro Bedankt dat je ons mee wilt helpen om onze website te verbeteren. We gaan je enkele vragen stellen over je favoriete spellen, en hoe je momenteel je spellen zoekt en vindt op onze website. Deze enquête neemt maar enkele minuten van je tijd in beslag.

A.1 Survey

A.1.1 Voorkeuren

1. Wat zijn je favoriete hoofdcategorieën op de Zylom website? Dit zijn de genres die je in het menu ziet staan, bovenaan onze pagina's, op dezelfde rij als Alle spellen, boven Mijn gegevens als je ingelogd bent. Kies alle categorieën die je regelmatig speelt.

- a. Kaart
- b. Puzzel
- c. Woord
- d. 3-op-een-rij

- e. Zoek en Vind
- f. Actie
- g. Time management
- h. Ik weet niet welke categorieën ik regelmatig speel
- i. Ik let niet op de categorieën die ik speel

2. Wat zijn je favoriete subcategorieën op de Zylom website? Dit zijn de genres die je ziet staan als je op een hoofdcategorie klikt, en er een vakje uitgevouwen wordt. Kies alle subcategorieën die je regelmatig speelt.

- a. Solitaire
- b. Poker
- c. Klassieke kaartspellen
- d. Mahjong
- e. Sudoku
- f. Legpuzzels
- g. Andere puzzelspellen
- h. Klassieke woordspellen
- i. Kruiswoordpuzzels
- j. Wisselspellen
- k. Bubble shooter
- l. Blokken klikken
- m. Blokken verbinden
- n. Avontuur (Zoek en Vind)
- o. Klassiek (Zoek en Vind)
- p. Sim/tycoon
- q. Arcade
- r. Andere actiespellen
- s. Boerderij
- t. Mode

3. Hoe vaak zoek je naar een specifieke speltitel?

- a. Altijd
- b. Vaak
- c. Af en toe
- d. Zelden
- e. Nooit

4. Kun je met onze huidige indeling en categorienamen de categorie of het spel vinden dat je zoekt?

- a. Ja, altijd
- b. Ja, vaak wel
- c. Soms wel, soms niet
- d. Nee, meestal niet
- e. Nee, nooit

5. Heb je moeite om eerder gespeelde spellen terug te vinden op onze website?

- a. Ja, want ik weet de naam vaak niet meer
- b. Ja, door de huidige indeling van de site kost het me veel tijd om het spel weer te vinden
- c. Soms lukt het me niet
- d. Soms, want ik kom vaak bij de downloadversie uit, terwijl ik op zoek ben naar de online versie (of andersom)
- e. Nee, ik kan mijn eerder gespeelde spellen eenvoudig en snel terug vinden
- f. Nee, want ik speel een spel vaak maar n keer
- g. Nee, want ik speel alleen spellen van de homepage
- h. Ik weet het niet, want ik heb nooit eerder spellen bij jullie gespeeld
- i. Anders, namelijk;

6. Gebruik je de optie in Mijn gegevens om vijf favoriete spellen te bewaren wel eens?

- a. Ja
- b. Nee
- c. Deze optie ken ik niet
- d. Ik log nooit in, dus kijk niet in Mijn gegevens

7. Gebruik je de bladwijzers of favorietenoptie in je browser (Internet Explorer, Firefox, etc.) om spellen gemakkelijker terug te vinden?

- a. Ja
- b. Nee
- c. Soms
- d. Ik zou dat wel willen, maar ik weet niet hoe het werkt

8. Gebruik je de geschiedenis in je browser (Internet Explorer, Firefox, etc.) om spellen gemakkelijker terug te vinden?

- a. Ja
- b. Nee
- c. Soms
- d. Ik zou dat wel willen, maar ik weet niet hoe het werkt

9. Laat je wel eens tabbladen met online spelletjes open in je browser (Internet Explorer, Firefox, etc.) om ze gemakkelijker terug te kunnen vinden?

- a. Ja
- b. Nee
- c. Soms
- d. Ik zou dat wel willen, maar ik weet niet hoe het werkt

10. Hoe lang duurt het gemiddeld voordat je een spel gevonden hebt dat je wilt spelen?

- a. Ongeveer 10 seconden
- b. Ongeveer 30 seconden
- c. Ongeveer 1 minuut
- d. Ongeveer 5 minuten
- e. Meer dan 5 minuten

11. Vind je het te lang duren voordat je een spel hebt gevonden om te spelen?

- a. Ja, het zou sneller moeten kunnen
- b. Ja, het zou gemakkelijker moeten zijn, zodat ik niet zoveel verschillende paginas door moet
- c. Nee, met de huidige indeling en categorienamen kost het me niet veel tijd
- d. Nee, ik neem juist de tijd om een spel uit te zoeken
- e. Het maakt me niet uit

A.1.2 Nieuwe zoekoptie

We hebben het plan om een nieuwe zoekoptie toe te voegen aan onze website, waarbij je bijvoorbeeld een spelnaam in kunt typen, en je dan de resultaten van je zoekopdracht op een rijtje ziet staan. Je kunt dan op een spelnaam klikken om meteen naar de infopagina te gaan en het spel te spelen.

12. Op wat voor manier zou jij willen zoeken in een dergelijke zoekfunctie? Selecteer alle mogelijkheden die jij zou gebruiken.

- a. Mogelijkheid om te zoeken op speltitel of een deel van de titel
- b. Mogelijkheid om te zoeken op de naam van een serie (bijv. Delicious, Dream Chronicles)
- c. Mogelijkheid om te zoeken binnen een beschrijving op de infopagina van een spel
- d. Mogelijkheid om te zoeken op de beschrijving van de omgeving of het onderwerp van een spel (bijv. de Middeleeuwen, gras, ballen)
- e. Mogelijkheid om te zoeken op genre
- f. Mogelijkheid om te zoeken op de naam van personages uit het spel (bijv. Emily, Flo)
- g. Mogelijkheid om te zoeken op de naam van de ontwikkelaar van het spel (bijv. PopCap, Alawar)
- h. Mogelijkheid om te zoeken op de prijs van het spel, of een aangegeven prijslimiet opgeven
- i. Mogelijkheid om te zoeken op verschijningsdatum
- j. Anders, namelijk;

13. Is er informatie die je momenteel niet op de website kunt vinden, die je graag zou willen toevoegen? Denk daarbij aan bepaalde informatie over spellen, of bijvoorbeeld onderwerpen die je niet via de Helpagina hebt kunnen vinden.

A.1.3 Achtergrond

14. Hoe vaak bezoek je gemiddeld de Zylom-website?

- a. Een paar keer per dag
- b. Een keer per week
- c. Een paar keer per week
- d. Een keer per week
- e. Een keer per 2 weken
- f. Een keer per maand
- g. Minder dan 1 keer per maand

15. Heb je wel eens een spel gekocht bij Zylom, of een FunPass-abonnement gehad?

- a. Ja, ik heb minimaal n spel gekocht
- b. Ja, ik heb momenteel FunPass
- c. Ja, ik heb in het verleden FunPass gehad, maar nu niet meer
- d. Ja, ik heb minstens n spel gekocht, en een FunPass gehad (of momenteel nog steeds FunPass)
- e. Nee, ik heb nooit een spel gekocht of een FunPass gehad

16. In welke leeftijdscategorie val je? We gebruiken deze informatie alleen voor onze statistieken.

- a. 20 jaar of jonger
- b. Tussen 21 en 40 jaar
- c. Tussen 41 en 60 jaar
- d. Ouder dan 60 jaar

17. Ben je een vrouw of een man? We gebruiken deze informatie alleen voor onze statistieken.

- a. Vrouw
- b. Man

Dit is het einde van de enquete. Bedankt voor het invullen, en nog veel speelplezier!

Appendix B

Search Engine comparison

This section contains the engines that I found the most references to when searching the web using Google.nl and DuckDuckGo.com starting from September 20, 2011. These references are forum questions, recommendations, lists made by others on the web and top results from the search engines.

B.1 Libraries

B.1.1 The old in-house developed code

1. Homepage: available on the intranet in a branch of the software repository
2. First release date: 2007
3. License + costs: Free, internal developed
4. Status: inactive
5. Support methods: Developer is no longer working at GameHouse
6. Minimum requirements: Java 6, defunct in Java version 7
7. Promoted features: It could be helpful to build upon this code, or reuse parts together with other packages or libraries. Has fuzzy search matching, sorting and displaying methods.
8. Missing features from demands list: auto-suggest, spell checking, language specific features, flexible behavior with the game types, provides inaccurate results and is obsolete, see the 'Testing' section below.
9. Programming Language: Java 6
10. Expandability: Can be expanded with any new code, just none provided by the developer himself.
11. Manageability: Same as any other internal code, via the internal repository.

B.1.2 Lucene

1. Homepage: <http://lucene.apache.org>
2. First release date: 2001
3. License + costs: Free, Apache License <http://www.apache.org/licenses/LICENSE-2.0/>
4. Status: Active
5. Support methods: Supporting companies are listed on the Lucene website
6. Minimum requirements: Java 6, defunct in Java 7
7. Promoted features: Apache Lucene(TM) is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform.
8. Missing features from demands list: statistical query logs, auto-suggest.
9. Programming Language: Java 6
10. Expandability: Several extensions available, amongst other, the packages listed below.
11. Manageability: Source code

B.1.3 Elasticsearch

1. Homepage: <http://www.elasticsearch.org>
2. First release date: 2010
3. License + costs: free, Apache License <http://www.apache.org/licenses/LICENSE-2.0>
4. Status: Active
5. Support methods: mailinglist, IRC channel (100+ visitors on Friday 23 September), bug tracking, documentation on the site,
6. Minimum requirements: JSON scripting, CURL, Java 6
7. Promoted features: Distributed, RESTful, Search Engine built on top of Lucene. Actual JSON document used to index the specific data. Fields in documents can have boost levels that affect scoring, analyzers can be used to control how text gets tokenized into terms, certain fields should not be analyzed at all, and so on... Elasticsearch allows you to completely control how a JSON document gets mapped into the search engine on a per type and per index level. One of the main features of Elastic Search is its distributed nature. Indexes are broken down into shards, each shard with 0 or more replicas. Each data node within the cluster hosts one or more shards, and acts as a coordinator to delegate operations to the correct shard(s). Re-balancing and routing are done automatically and behind the scenes. Issuing queries is a simple call hiding away the sophisticated distributed based search support Elasticsearch provides. Search can be executed either using a simple,

Lucene based query string or using an extensive JSON based search query DSL. Search though does not end with just queries, facets, highlighting, custom scripts, and more are all there to be used when needed. log4j logging, query logging possible.

8. Missing features from demands list: NoSQL instead of SQL
9. Programming Language: Java 6
10. Expandability: modules available on the site.
11. Manageability: Source code

B.1.4 Solr

1. Homepage: <http://lucene.apache.org/solr/>
2. First release date: 2004
3. License + costs: free, Apache License <http://www.apache.org/licenses/LICENSE-2.0>
4. Status: Active
5. Support methods: On the Solr website, there are dozens of companies listed that give support. <http://wiki.apache.org/solr/Support>
6. Minimum requirements: Java 6, defunct in Java 7, runs in a servlet container like Tomcat.¹ Uses Lucene.
7. Promoted features: Advanced Full-Text Search Capabilities, Optimized for High Volume Web Traffic, Standards Based Open Interfaces - XML, JSON and HTTP, Comprehensive HTML Administration Interfaces, Server statistics exposed over JMX for monitoring, Scalability - Efficient Replication to other Solr Search Servers, Flexible and Adaptable with XML configuration, Extensible Plugin Architecture,
8. Missing features from demands list: none
9. Programming Language: Java
10. Expandability: Extensions to the Lucene Query Language
11. Manageability: Comprehensive HTML Administration Interfaces, can be used without programming.

¹Tomcat can give problems with internationalized texts. UTF-8 conversion problem. Alternative is needed.

B.1.5 Hadoop Search, HSearch

1. Homepage: <http://bizosyshsearch.sourceforge.net/>
2. First release date: 2010
3. License + costs: Apache License Free <http://www.apache.org/licenses/LICENSE-2.0>
4. Status: Active
5. Support methods: Website for bug reports, forum (not actively used), documentation is hidden between the API descriptive files.
6. Minimum requirements: depends on HBase, Java (version not mentioned)
7. Promoted features: Multi-XML formats, Record and document level search access control, Continuous index updates, Parallel indexing using multi-machines, Embeddable inside application, A REST-ful Web service gateway that supports XML, Auto sharding, Auto replication, log4j logging: query logging possible.
8. Missing features from demands list: It works on HBase, a NoSQL database is a dependency. Data can be exported from Microsoft SQL Server with the tool sqoop, but then it should be exported back to the server again. If this does not happen, the demand for statistics from MSSQL is not met. This importing and exporting, no matter how fast the data transfer is, will create overhead. With fields from a large, frequently updated database this can be tedious.
9. Programming Language: Java 1.6
10. Expandability:
11. Manageability: Nothing mentioned outside the promoted features.

B.1.6 Xapian

1. Homepage: <http://xapian.org>
2. First release date: 1999
3. License + costs: free, GNU GPL v2 licensed. <http://www.gnu.org/licenses/gpl-2.0.html>
4. Support methods: paid consultancy at 3 companies , including development on demand. Free tutorials, documentation, mailing-list.
5. Status: Active
6. Minimum requirements: A 'C' compiler and Make program on a Unix environment
7. Promoted features: Used by Tweakers.net. BM25 Weighting Scheme Collapsing, Database Replication, Faceting, Indexing, PostingSource, Query Parser, Remote Backend, Serializing Queries and Documents, Sorting Results Spelling Correction, Stemming Algorithms, Synonym Support, Value Ranges,

8. Missing features from demands list: not fully in Java . Auto-suggest previous searches.
9. Programming Language: C++ with bindings to allow use of the Java syntax
10. Expandability: tag library for search results. XML source configuration file. More file formats can be searched (next to text html etcetera) with use of preparators. These preparators are basically parsers for data formats.
11. Manageability: Source code

B.1.7 Sphinx

1. Homepage: <http://sphinxsearch.com>
2. First release date: 2001
3. License + costs: free, GNU GPL v2 licensed. <http://www.gnu.org/licenses/gpl-2.0.html> Paid support.
4. Support methods: free community support through a Forum, Wiki and IRC. Well documented. Support ranging from \$1500 for a 3 days response to \$15000 a year for phone and Skype support. Also consultant services available for shorter support durations and developers can be hired to create features.
5. Status: Active
6. Minimum requirements: A 'C' compiler and Make program. Any database can be used through ODBC and it is written in C for *nix, Windows and Mac.
7. Promoted features: high indexing and searching performance; advanced indexing and querying tools (flexible and feature-rich text tokenizer, querying language, several different ranking modes, etc); advanced result set post-processing (SELECT with expressions, WHERE, ORDER BY, GROUP BY etc over text search results); proven scalability up to billions of documents, terabytes of data, and thousands of queries per second; easy integration with SQL and XML data sources and SphinxAPI, SphinxQL or SphinxSE search interfaces; easy scaling with distributed searches. Used by large websites such as Slashdot.org.
8. Missing features from demands list: Not written in Java, but the query demands are natively met. Tag library.
9. Programming Language: C++, however there is a Java library named SphinxAPI that looks very useful.
10. Expandability: plug-ins available
11. Manageability: Source code

B.1.8 Regain

1. Homepage: <http://regain.sourceforge.net/>
2. First release date: 2004
3. License + costs: free LGPL licensed <http://www.gnu.org/licenses/lgpl.html>
4. Support methods: A forum, well documented help section.
5. Status: Active
6. Minimum requirements: servlet engine like Tomcat.
7. Promoted features: built on Lucene, Tag library: this is the first in the list that mentions it.
8. Missing features from demands list: query logs
9. Programming Language: Java
10. Expandability: standalone crawler, preparators enable expansion of type of files that can be searched
11. Manageability: Source code

B.2 Search engine services(Software As A Service)

These are the services that don't run on a server locally, but that are hosted elsewhere, indexing the company's website for a fee. The database and programming language requirements are not relevant in this case. I started looking at these services, but during the course of comparing search engines I soon found several candidates that already suit the needs and are free. The solutions underneath are quite expensive for large data queries and therefor don't seem like a good alternative to a regular local search engine implementation. Another issue is that when the source of the engines is available, it is easier to use existing Java code for the website together with the program. SAAS is not flexible in that sense.

B.2.1 Google site search

1. Homepage: <http://www.google.com/sitesearch/>
2. Costs: 2000 Euro annually
3. Support: Support by email, advanced developer documentation and online training.
4. Modifiable behavior: yes, users can sort on tags that appear above results.
5. Manageability: customization via a simple web interface

6. Promoted features: Word synonyms, fast indexed searching and sorting results biased towards certain categories (if they should be promoted). It also provides statistics on how many queries were done and it allows category search. Multiple language queries.
7. Missing features: complete data on which terms were sought after. It can show the most popular searches, and usage statistics. Interoperability with Java code.

B.2.2 Yahoo Build your Own Search Service

1. Homepage: <http://developer.yahoo.com/search/boss/>
2. Costs: Query based pricing would come down to \$1600 per month based on the statistical minimum of 2 million searches.
3. Support: A forum for general questions, as well as billing and service issues. Documentation in abundance.
4. Modifiable behavior: results can be re-ranked.
5. Manageability: Customization via a dashboard that also shows daily usage statistics.
6. Promoted features: Unlimited Queries Per Day (See Pricing Chart), No Restrictions on Presentation, Re-Ordering Allowed, Blending of Proprietary and Yahoo! Search Content Allowed, Access to a Limited web data set, Limited Yahoo! Brand Attribution, User Authentication Security, Self-Serve with Credit Card Authentication at Sign-up,- Yahoo! Search Advertising, Access to Usage data and Billing, Extensive Support and Documentation.
7. Missing features: A disappointment is the need to add the Yahoo logo and a statement if implementing the search engine on your website. Very expensive compared to alternatives.

Appendix C

Documentation

This is the literal documentation written for the intranet.

C.1 Lucene Search package

This page is about the Lucene Search package. It is installed along with other related packages in your external-libs folder. The version this page relates to is 3.4.0 Data is added to the generated index by documents. A document is comparable to a row in a database. It contains the field names, the values and things like if it has to be analyzed or stored.

- A document is a sequence of fields.
- A field is a named sequence of terms.
- A term contains a string like a game name.

I created an index in RAM named RAMgameIndex, it is also possible to make an index on the hard disk for perhaps testing purposes. The indexes are built in the class `com.zylom.website.search.LuceneIndexManager`, the searches, filters and sorting are done via the class `Search`. To read more about how Lucene works from building the index to parsing a query, visit http://lucene.apache.org/java/3_4_0/demo2.html For a simple implementation see: <http://www.lucenetutorial.com/lucene-in-5-minutes.html>

C.1.1 Document Fields

These are the defined search fields: `private final String`

```
GAME_FIELD_NAMES={"gamename","gamedescription","gamehelp",  
"languageID","category","skinnedgameID","gameSetID",  
"launchdate","gametype"};
```

Searching in a certain field is done by means of for example `gamename:Plants`. This searches only the game titles for Plants. To search in the `gamename` and `gamedescription` fields: `game-name:Plants AND gamedescription:shoot`

C.1.2 Managing the Index

Because there is no check for duplicate data in the index, this is taken care of in the `LuceneIndexManager` class. To easily test queries on the index the standalone tool Luke can be used, but then you have to create indexes on the hard disk. Syntax for creating a Directory (index) on the hard disk: `Directory gameIndex = new FSDirectory(new File(path));` Get the tool Luke from <http://code.google.com/p/luke/>. You can also browse every individual document there, see term statistics and even modify data.

C.1.3 Filtering and Sorting

Filtering starts in `searchfilter.JSP`, goes through the servlet `Filter.java`, calls `LuceneSearch.filterGames()`, then it is sent to `LuceneSearch.findGames()` like any other query. The servlet takes the user's chosen filters as requestparameters from `searchfilter.JSP` and sends them to `LuceneSearch.filterGames()` for filtering. It takes the previous query and all the chosen options like categories and a launch date range and formats a String accordingly. Sorting is done in `LuceneSearch.sort('Nameofsorttype')()`. In the case of the launch date and alphabetically, a given `ArrayList` is sorted via the `GameComparator` class. The old class has been adjusted to work with downloadable and online games. `sortRelevance()` simply does a new query. `sortPopularity()` first divides the games into a Downloadable and Online games list. Then each uses the relevant version of `getGamesList()` and applies `gamesList.retainAll('earlierGameSearchResults')` and concatenates the two sorted gamesLists into one.

C.1.4 Searching

`LuceneSearch` as its name implies, handles all of the searching. It starts in the Search servlet, where first the `LuceneSearch.formatQuery()` method is called. This removes excess spaces around the query and filters out any special characters except quotes. It also adds 'AND' or 'OR' between words. Then the journey continues to `LuceneSearch.filterGames()` that retrieves the data from the index and retrieves the corresponding games, storing them in an `ArrayList`. Via the Search servlet the data ends up in `searchresults.jsp`. `Searchresults.jsp` is a refurbished `gamecategory.jsp`, without the category things, changed layout and it shows both types of games. If the first game in the list is an online game, the downloadable version is sought, and vice versa. To avoid duplicate entries (downloadable game of 'Delicious' found in the `ArrayList`, after the online version already fetched and displayed 'Delicious deluxe') the other version of a game gets removed from the `ArrayList`.

C.2 Lucene Query Syntax

Searching is done with terms and phrases. Terms are single words, phrases are word groups in quotes. Details are at http://lucene.apache.org/java/3_4_0/queryparsersyntax.html

C.2.1 Enabled by default

Wildcards

? and * are allowed. Queries are not allowed to start with it.

Fuzzy Searches

Are done with ~ in the end of the word. An additional (optional) parameter can specify the required similarity. The value is between 0 and 1, with a value closer to 1 only terms with a higher similarity will be matched. For example: Emil~0.8 The default that is used if the parameter is not given is 0.5.

Proximity Searches

Words in a certain distance from each other. The following will search those words within a 20 word distance from each other.

```
"Chicken throw"~20
```

Range Searches

Finds documents that fall between a given range and sorts alphabetically. Inclusive range queries are denoted by square brackets. Exclusive range queries are denoted by curly brackets.

```
mod_date:[20020101 TO 20030101]
gamename:{Adventure TO Zombies}
```

Boosting Terms or Phrases

Can be done with the caret ^ symbol. For example if a game is part of a promotion, the relevance rate is boosted like this:

```
"Delicious Emilys true love"^4 or Delicious^4
```

This makes it four times more relevant. Floating point numbers are also allowed. See the section 'Search Term Rating Methods' for more boosting methods.

Field Grouping

Uses parentheses to group multiple clauses to a single field. To search for a title that contains both the word "return" and the phrase "pink panther" use the query:

```
gamename:(+return +"pink panther")
```

or the format

```
(Mens OR niet) AND erger
```

The dash excludes documents that contain the term after the - symbol. bookworm -"word game"

Boolean Searches

Boolean searches clearly follow the Java logic.

```
Garden AND farm
gamenname:Garden AND gamedescription:pick
```

The OR operator is available and `Plants +Zombies` evaluates to "must contain Zombies, with optional Plants". NOT can be used with multiple terms only. `Mortimer NOT graveyard`. It fails when using one term. NOT can also be replaced by `!`, AND by `&&` and OR by `—`.

Special Characters

```
+ - && || ! ( ) { } [ ] ^ " ~ * ? : \
```

They can be escaped by a forward slash `/`, but are already filtered out in the implementation, when reading the user input from the text field. The only symbol not filtered are quotation marks.

C.3 Extra libraries

Find the current additional libraries at <http://repo1.maven.org/maven2/org/apache/lucene/>
Implemented: Analyzers: Every language used on the website(in October 2011) has its own term analyzer added. It determines which words are most likely relevant to the rating system. `SpellChecker` is the class for spell checking. Only one for all languages.

C.4 Search Term Rating methods

For all the details, see http://lucene.apache.org/java/2_9_1/scoring.html

C.5 Lucene Lingua Operandi

1. `tf` = term frequency in document = measure of how often a term appears in the document
2. `idf` = inverse document frequency = measure of how often the term appears across the index
3. `coord` = number of terms in the query that were found in the document
4. `lengthNorm` = measure of the importance of a term according to the total number of terms in the field
5. `queryNorm` = normalization factor so that queries can be compared
6. `boost (index)` = boost of the field at index-time
7. `boost (query)` = boost of the field at query-time

Lucene allows influencing search results (for example, promos) by "boosting" in more than one level:

- Document() level boosting - while indexing - by calling `document.setBoost()` before a document is added to the index.
- Document's Field level boosting - while indexing - by calling `field.setBoost()` before adding a field to the document (and before adding the document to the index).
- Query level boosting - during search, by setting a boost on a query clause, calling `Query.setBoost()`.

C.6 Tags

The new package `com.zylom.website.tag` has been added that contains the classes `Tag` and `TagManager`. The format works after example of the other base classes and Managers: implementing the `PersistentStoreable` interfaces, getting data from the database, equivalent getters and setters etcetera.

C.7 Problems to be anticipated

Adding a language, (no failure expected)

When implementing a new language and assigning it to a game, it will use a language independent analyzer by default. To do this operation more accurately go to <http://repo1.maven.org/maven2/org/apache/lucene/lucene-analyzers/> and add the analyzer class to the external libraries. Add the analyzer to

```
com.zylom.website.search.LuceneBuildManager public static final List <Analyzer> analyzers in the correct location. So if the new language is simply highest-id-in-database + 1: add it to the end of the list. The analyzers tokenize descriptions into individual words and allow word stemming. This tokenizing should not be done on items that need an exact match like date strings!
```

C.7.1 Duplicate entries

When making changes to the indexing process, please be aware that the index does not detect duplicate entries.

C.7.2 Updating existing documents

The only updating function is `indexWriter.updateDocument(Term,Document)` It removes and re-adds the document. Note that `Document` and `Term` are object types, not meant to just be descriptive. The constructor is `Term(String fieldname)` or `Term(String fieldname, String textToMatch)` Because this would match(update) every language and you can't do the update with several `Terms` as parameter, there is an `updateEntry()` function in `LuceneIndexManager`. Thanks to a switch, this can be used for several purposes like adding tags to games. Because

finding a Document to be replaced requires several arguments (language, wordtomatch) a separate `getDocument()` function is made in `LuceneSearch`.

C.7.3 Multiple entries per field

You can have multiple entries per field, by adding the field/value several times, not by using a comma like `tag="restaurant","Emily"`. You might for example want several tags per game. In the index you need to add the following in a loop

```
GAME_FIELD_NAMES[8]=tag
```

Or of course add them all in the same String like in

```
GAME_FIELD_NAMES[8]=all tags
```

C.7.4 Irrelevant game results

You probably forgot to use the query formatter, which adds braces and boolean operators to the search term.

C.7.5 Discrepancy in the documentation

The documentation on the official website is on some points obsolete. I for instance read that there was no update function for Documents at all. Please bear this in mind and read the API documentation instead.

Bibliography

- [1] Basham, Bryan. Sierra, Kathy. and Bert Bates *Head First Servlets & JSP* [2004] O'Reilly, First Edition.
- [2] Croft, Bruce W., Metzler, Donald. and Trevor Strohman *Search Engines, Information retrieval in practice* [2010] Pearson, international edition.
- [3] Grood de, Derk-Jan *TestGoal* [2008] Sdu Uitgevers, First edition.
- [4] Krug, Steve *Don't Make Me Think!* [2000] New Riders, First Edition.

List of Figures

2.1	Company structure	8
3.1	Search design given for the final product.	9
3.2	Results page and filter design given for the final product.	10
4.1	The frequency of visits given by players.	13
4.2	Information on bookmark usage given by players.	13
6.1	Class diagram of some of the games related classes	21
6.2	Database class diagram	22
7.1	Class diagram of the search related classes.	24
7.2	Class diagram of the Search Servlet package.	26
7.3	Class diagram of the Tag package.	28
8.1	Function chart, comparing speed of indexes.	30

List of Tables

4.1	Grouped suggestions.	14
7.1	Search process use case diagram.	23