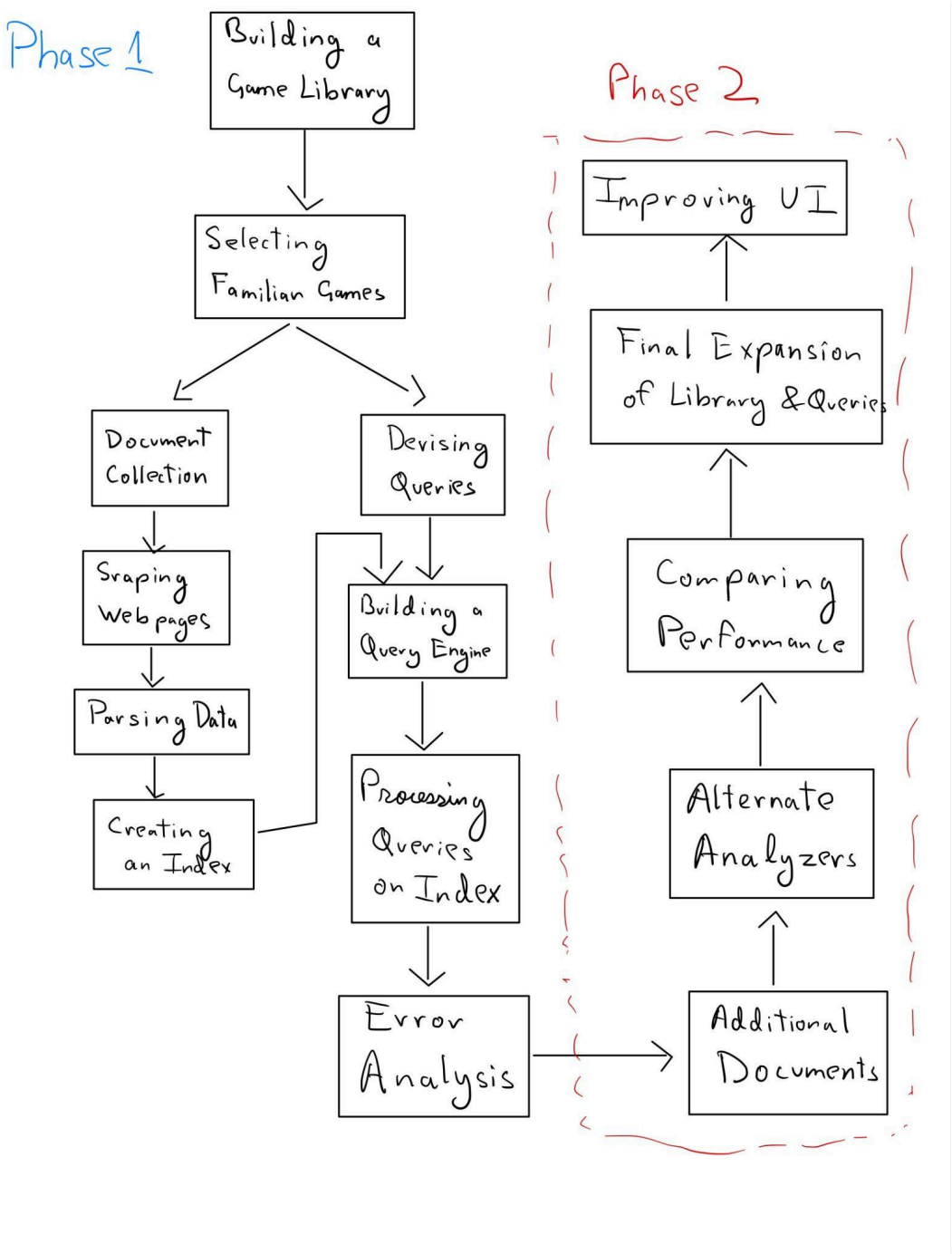
CSC 483 Project Report - Gamer Archives

Team:
Priyansh Nayak
Mahesh Magendaran
Alberto Andres Sanchez
Alex Scott Gable
Osama Alzahrawi



Description of Code and Directory

The central file to run is GamerArchives.java which hosts the search engine. It runs on our index called norgindex, and performs an analysis using a CustomAnalyzer catered towards stop word removal, stepping, case insensitivity, and standard tokenizer, and uses the BM25Similarity to compare documents when a query is processed.

If the index called norgindex is not present in the directory, then run NorgIndexBuilder.java to create it first. Ensure that the index is in the parent directory. Maven should have precompiled all the required files. NorgIndexBuilder.java uses the documents present in the resources folder, by assigning the docName as the name of the folder a document is in, which represents the name of a video game. There are a total of 967 documents spanning 100 different video games.

A sample run of the option is present in a file called output.txt in the git repository. It showcases the results of each query and which bucket it ends up in and prints out the statistics of those queries as well as performance metrics.

Generally, we established a 64% guaranteed success rate, which boosts up to 87% when the result is in the top 5 retrieved documents, and 93% when it is in the top 10 retrieved documents. This leaves a mere 7% failure failure rate.

We have a bunch of Python files that do scraping and query building for us. They are as follows:

MochiThreadRunner.py is responsible for scraping reddit links provided in a text file called redditLinks.txt present in the resources folder. Each line has one line.

NorgQuestionGenerator: Looks into a text file called queries.txt and produces questionBank.txt by pairing up queries and expected answers in a random order.

NorgWikiScraper: Scrapes wikipedia pages using article names present in library.txt

The performance metrics and statistics from the sample run are elaborated more in the section about Performance Metrics. Additionally, a sample run is present in output.txt

An example run would print this out for you:

> Welcome to Games Archive!
> Which mode would you like to check?
>
> 1. Run pre-built queries
> 2. Enter your own query

Press 1 to run our collection of 330 queries, or press 2 to have fun with your own queries!

Core Implementation

For this project, we utilized Lucene as our IR system. Still, instead of indexing the Wikipedia collection normally like Watson, we decided to build our collection of video game-related documents primarily using Reddit to create a game library that fits our needs.

The standard Watson prototype was supposed to be designed for general knowledge trivia, but ours is catered towards video games (ones that we have some personal experience with) and interacting with them based on information fellow gamers are likely to ask. Please note that the answers would all be names of games since that is what we targeted our engine to be about, as we want to understand what types of questions are usually asked in game communities.

Parsing Documents for Index Building

For term preparation, we utilized Reddit as our main source of document collection since reddit threads involve the most discourse surrounding video games. We initially relied on the StandardAnalyzer to parse our documents, and we built our index such that one document object would have all of the various documents about a certain game compiled together as its content.

However, to increase performance, we adapted the EnglishAnalyzer to improve the filtering process of the terms, and soon moved on to a CustomAnalyzer to add additional features we wanted to take into account, improving it to support stop word removal, stemming, and case-insensitivity.

As a result, we went through at least 4 different versions of Analyzers. Later on, we improved indexing to have each document we collected be its own entry in our index instead of compiling multiple into one singular entry.

Issues with Document Collection

Because of the nature of our project, we decided to use Reddit forum threads as content to build a collection instead of Wikipedia. One of the issues we ran into when targeting specific forum threads was the existence of hidden comments for aesthetic purposes, which made the conversion into text file data difficult.

For this, we utilized PRAW, a Python wrapper that aided heavily in scraping Reddit for data and allowed us access to all comments. Another issue we encountered was emojis and text art used primarily for meme reactions on comment threads.

Our Custom Analyzer was designed especially to filter out these issues. One other issue that we had was how our program was retrieving the wrong queries despite having rather accurate documents for the program to retrieve the right document.

For instance, some queries such as the voice actor for Mikasa voiced in what game? Would not return Nier Automata. The fix for this was to tamper with the query itself. In this case we changed the query to Yui Ishikawa (name of the voice actor) voiced for what game and a new reddit link that had Yui Ishikawa in the thread itself was added. This would solve the issue allowing it to retrieve the document we want.

Query Descriptions

The primary approach to generating queries involved each team member running a scraper on Reddit sites related to the game. The number of queries was distributed among team members to meet specific quotas for collecting game information (documents). The way we divided the queries was based on the games that each team member themselves has played as familiarity plays a big role when thinking of specific game related queries.

To delve deeper into how the scraper was used, consider this example: To gather queries for the game Honkai Star Rail, a team member would first visit the Honkai Star Rail Reddit page. Each team member was responsible for creating queries for the games assigned to them. They would then type their questions on a Google Doc named 'doc collections,' which typically contained an average of 3-5 questions per game.

> Honkai Star queries from the google doc:
>
> Honkai Star Rail
> Which game are Kafka, Huohou, Ruan Mei, and Black Swan a DOT team from?
> What is the most popular game that Prydwen makes a tier list for?
> Where is the game mode MoC from?
> What game has a character called Firefly?

Continuing with the example of Star Rail, after entering the Reddit page for the game, the team member would search for Reddit threads related to that game which could answer the queries listed in the 'doc collections.'

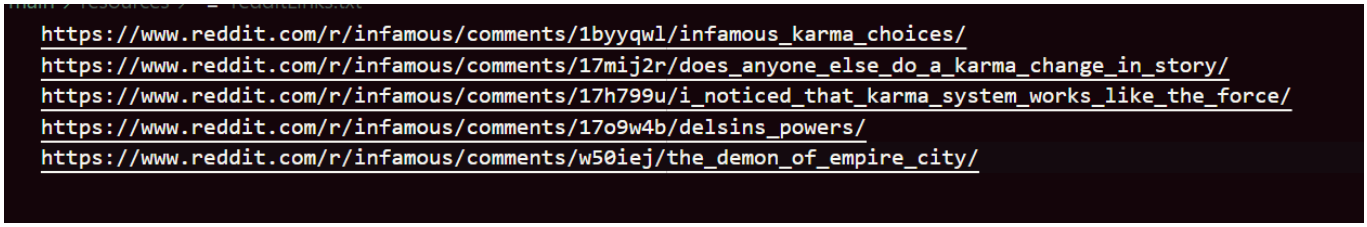
The links to these threads would be added to the 'redditlinks.txt' folder, which served as the source for the scraper to retrieve our documents. (The target was approximately 10 Reddit links per game.)

The scraper would then prompt the user for a filename, and it would create a file in the directory named after the user's input. This file would contain all the scraped data from the Reddit links entered in the TXT file.

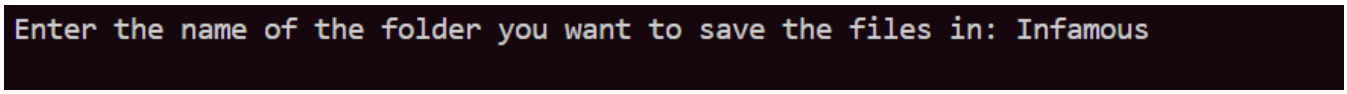
Returning to Honkai Star Rail as our example, if someone typed a query on the Google Doc such as 'Where is Acheron from?' the team member would then search for relevant Reddit documents to be scraped.

Images to show how the scraping worked:

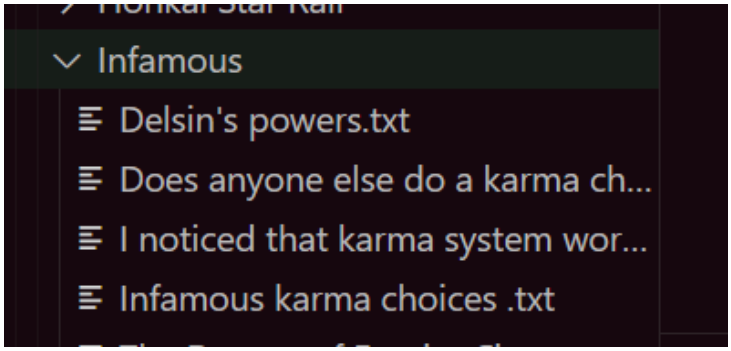
Adding links to the redditlinks.txt file



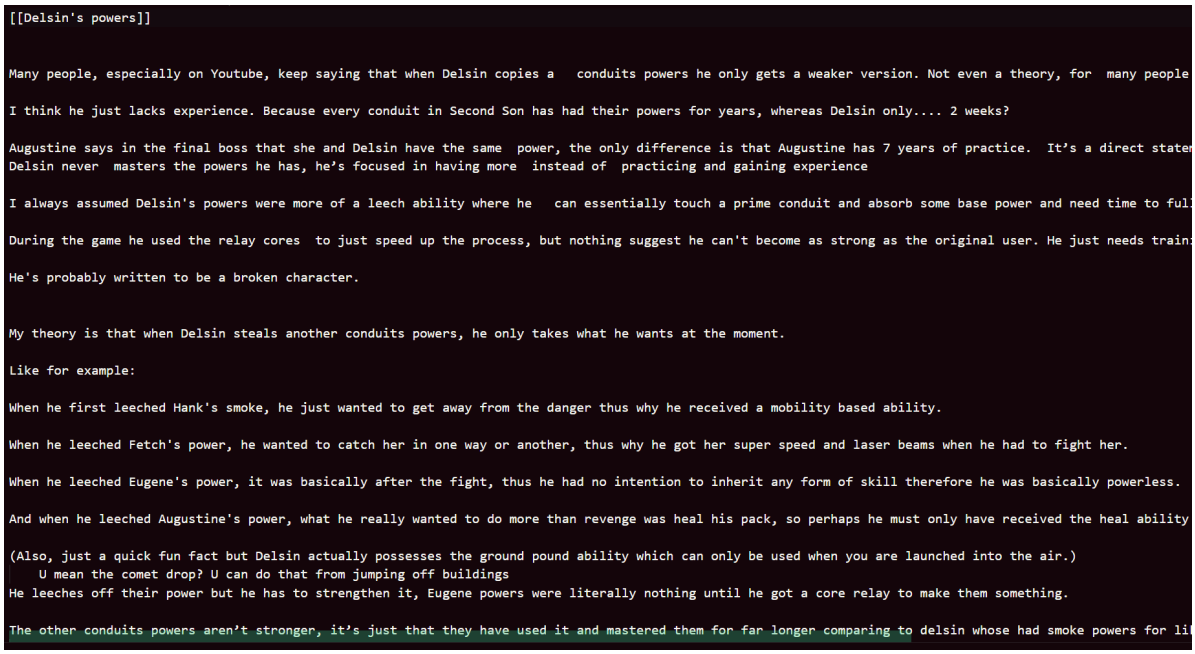
Enter name of folder



Scraper adds it to directory



Example of scraped data



Measuring Performance

We chose to utilize the metrics P@1, P@5, P@10, and MRR.
There's a reason why I choose to talk about 4 different metrics, but the primary one I will prefer is P@5. Videogames tend to utilize a lot of similar concepts and tropes that it becomes second nature to bring up certain topics when it comes to discourse on videogame related forums. As a result, during testing a lot of the queries resulted in misses when we didn't expect them to. Initially, we moved toward targeting specific types of reddit threads to suit those, and included wikipedia sites for about half of the library.

Still noticing unreasonable misses, we decided to investigate how far off the query's retrieved results were, and introduced two more buckets signifying to see if the expected answer was in the top 5 documents retrieved, or in the top 10 documents.

Upon making that split, we realized that there were a lot of queries whose retrieved answers that match the expected answers ended up in the top 5 buckets. The reason being the occurrence and usage of the terms mentioned in the query occurring more in a different document, usually ones that were much larger than the game the query was targeting.

Observing the Mean Reciprocal Rank provided a final score of 0.73, which told me more about how beyond the correct hits, the buckets of top 5 an top 10 impacted just how much larger documents skewed our results.

As a result, here are the final details regarding the final stats based on our CustomAnalyzer and the BM25Similarity, with the primary metric being P@5, and MRR for some info:

----------Stats----------

Total Queries: 330
Correct Answers: 210
Within Top 5: 79
Within Top 10: 19
Incorrect Answers: 22

----------Performance----------

Precision at 1: 0.6363636363636364
Precision at 5: 0.8757575757575757
Precision at 10: 0.9333333333333333
Mean Reciprocal Rank: 0.7317556517556518

For some context, this is the very first version of our engine using a StandardAnalyzer and the default similarity on the full library:

----------Stats----------

Total Queries: 330
Correct Answers: 201
Within Top 5: 76
Within Top 10: 20
Incorrect Answers: 33

----------Performance----------

Precision at 1: 0.6090909090909091
Precision at 5: 0.8393939393939394
Precision at 10: 0.9
Mean Reciprocal Rank: 0.702497594997595

Error Analysis

How many questions were answered correctly/incorrectly?:

When we tested the first version of the gamer archives program, we found that out of 100 sum queries, about half of the queries were wrong. Upon expanding our document library and adjusting how the queries are interpreted this allowed for answers to be found more consistently, we saw great improvement. We also settled on BM25Similarity after encountering issues with the classic TF-IDF variant and BooleanSimilarity of Lucene.

We now have over 900 Documents and information from 100 different games. As the amount of games increased, we saw more false positives because some games would have similar gameplay features and hence similar contents in certain documents.

This occasionally led to some false positives, but overall our system accuracy had increased as we continued to fix bugs we found. Oftentimes a bug would be found when a new game was added and it would either not be found at all, or it would be found from questions that were not intended form it. We then were able to accomplish having out of 330 queries only 22 incorrect answers.

Why do you think the correct questions can be answered by such a simple system?:

The questions that this system can answer work because we can compare the contents of the question with the contents that are found on reddit or other sites. The reason we choose to use reddit is because it is a hub of discussion that is often filled with questions about the games that we picked. So when we compare our questions to the discussions that are held on reddit, we

can find the answers based on what game we have categorized the document to relate to. This does rely on having a large amount of documents and a relatively even amount of documents for each game, so that larger documents do not affect the score for certain queries so heavily that it would outweigh a potentially correct answer.

What problems do you observe for the questions answered incorrectly?:

The script assumes that all Reddit links follow a standard format. However, there may be special cases where the format deviates, causing the script to fail or produce incorrect results. Also, if the Reddit threads contain references to game names in a specific term or use shortcuts, the script may misinterpret them. For example, if a game is mentioned in a comment in a specific way, the scripter may miss it. Reddit users may use humor, like emojis, which can make it difficult to understand the intended meaning or topic.

Try to group the errors into a few classes and discuss them.

Query errors: Query errors occur when there are vague queries; they may be ambiguous and general, making it challenging for the query engine. Queries that contain terms commonly associated with multiple games. For example, “open-world” and “first-person shooter” Also, some Reddit users use slang or shortened versions of words or names that our system would not pick up.

Matching errors: Errors where there was nothing wrong with the document nor the query, but for some reason the document was not matched with the correct answer. These errors were only prominent when our system was still in development. It usually happens now because of a really large document that has similar terms, giving a false positive for a game that it should not.

Document errors: document errors occur when there are issues with the indexed documents themselves, affecting their accuracy for the user's queries. For example, many of our Reddit documents had hidden comments or emojis that would cause errors.