**Title:** Wikipedia Recommender System

**Authors:** Samuel Janas 151927, Michał Skrzypek 151766

**Date:** 05.11.2023

1. **Goal**

Our project aims to create a recommendation system for similar articles based on users' previously visited articles in an effective and ethical way. The system comprises several key components: web scraping and crawling, text preprocessing, content similarity calculation, and article recommendations.

1. **Data Collection (Crawling and Scraping)**

The data collection process involves scraping content from fandom website about computer game Terraria. We choose this fandom Wikipedia because Terraria is superb.

**config.py**

**<!-- paste config file here>**

Configuration parameters such as base, start URLs and the number of articles that we want to scrape are provided. We defined sleep rule to avoid excessive and rapid requests that could overload the website's server, resulting in getting our IP address banned. Parameter MAX\_DEPTH limits the depth of web page traversal during the web scraping process.

**scraper.py**

With global variable VISITED\_LINKS we keep track of already visited links so none of them overlap. We store scraped data in list called articles. Functions fetch\_robots\_txt, parse\_robots\_txt, can\_visit\_url retrieve content of the file robots.txt then parse it and extract “Allow”/”Disallow” rules and finally determines whether the link can or cannot be visited. This part ensures that we crawl through the website ethically and for instance we don’t scrape data about users. With get\_links\_from\_url we extract all links from a given URL and then we have next websites to visit. We consider links starting with predefined BASE\_URL only. The scrape\_content\_from\_link scrapes the main content from a given URL. It sends a request to the provided URL, parses the HTML content, and searches for the main content, assuming that it is located in a <div> with the class "mw-parser-output. The dfs\_crawl function performs a recursive depth-first traversal of the website. If the content of a page is successfully scraped, it is appended to the articles list.

1. **Preprocessing: Lemmatization**how and why here
2. **Similarity Calculation & Recommendations**

We use “Term Frequency-Inverse Document Frequency” because this technique takes into account two crucial factors:  
- how often a term appears in a document  
- how unique a term is across documents

The next step is converting the article content into TF-IDF vectors and then matrices. We calculate the similarities with cosine-similarity formula. It calculates the cosine of the angle between vectors.

**# The resulting recommendations are based on the content of the articles.**

**-** show some charts about which words occur most often.

**-** show 1-3 recommendationsmathermaticaly (e.g words that are similar here and there)

- 1-2 recommendations knowledge-based (ask sami)

When you explain anything use snipptets of code. Use markdown ffs.