G.O.A.T.

Go On And Try: A Media Recommendation Service

**Team Members:**

* Peyton VanHook
* Peyton White
* Sean Hopkins

**Abstract**

Goat will be a desktop-based media recommendation service that provides recommendations for new media to users based on user preferences. The service will aggregate listings of video games, movies, television programs, and books, and will use natural language processing to distill data on media properties into a set of common tags. These tags will represent genre, tone, theme, content ratings, critical aggregate scores, and more. The key to Goat’s recommendation service will be our search algorithm, which, when provided with tags, will find media properties in it’s database that are most associated with the tags provided.

As we develop Goat, we plan to add in further user profile functionality beyond the recommendation algorithm, allowing users to scroll through a feed of tags and tag sets that the user most commonly looks for, and allowing them to provide user feedback data to supplement our database’s aggregated data.

**Description**

Data is sourced from a number of pre-created databases and associated API’s. For movie data, our database is sourced almost entirely from “The Movies Dataset”, uploaded to kaggle.com by Rounak Banik, a student at IIT Roorkee in Roorkee, India. The dataset, licensed in the public domain, contains data on 45,000 movies produced up until 11-9-2017. The data in this dataset includes, among other things, movie titles, IMDB database identifiers, plot overviews, user review scores, critical aggregate scores, genre information, and associated keywords. Additional data, such as movie and television cover images, will be pulled from IMDB, TMDB, and OMDB’s APIs.

For books, we will utilize the two pre-created databases. The first is “Goodreads-books” uploaded to kaggle.com by user Soumik, a student at Jalpaiguri Government Engineering College in Jalpaiguri, India. The second is “goodbooks-10k” uploaded to Github by user zygmuntz. These databases contain listings on 10,000 books, including titles, ISBN, user review score and count, and other data. Additional information will be sourced from Goodread’s API and Google Book’s API.

For video games, we will source our data from the RAWG API. The database contains more than 350,000 games for 50 platforms. RAWG includes listings such as tags, genres, developers, publishers, creators, official websites, release dates, and Metractritic ratings. It has free commercial use for startups and hobby projects as long as there are no more than 100,000 monthly active users or 500,000 page views per month.

Our keyword data used by the recommendation algorithm will be sourced from databases where it exists, and generated from plot summaries on the web where it cannot. For our movies and television listings, keyword data and plot summaries are included in the source database. So the recommendation algorithm will use the keywords already included in the database as well as tags generated from the included plot summaries. For cases where we use plot summaries, we will source this information from a Wikipedia archive sourced from Wikipedia’s export utility.

We plan to store each media item in a single database, and each item will link to a more specific database that contains details that are not shared between media types. For example, the common database will contain the item’s unique ID in all databases, the item’s media type, the item’s title, the item’s array of associated keywords, the item’s summary, the item’s publishing date, a link to a cover photo in a photo cache, and a link to a specific database that contains media specific information (for video games, for example, what operating system/platform they run on, what game engine/platform they use, 3D or 2D, etc).

The database will run and be stored in a mySQL server. The software used to control the database will be PHPMyAdmin 4.9.4.

The core of Goat will be the recommendation algorithm. The recommendation algorithm will find a listing of media properties of any type that most closely match the tags that the recommendation algorithm is given. The recommendation algorithm will then return the results ordered by how many tags were shared between items. The search algorithm itself will be written in Java and be part of our local software.

The tags we utilize will be, as much as is possible, pre-programmed tags sourced from databases along with release dates, critical and user aggregate review scores. However, in the future we would like to expand this to include an NLP-based keyword extraction system, so that we can take the text from plot summaries and critical/user reviews to determine tags procedurally. Ideally, this system would utilize Microsoft Azure Cognitive Service’s key phrase extraction text analytics utility or an equivalent.

In the initial prototype, the front-end user interface will be completely focused on the recommender system. The user will input a media title, ISBN, or IMDB ID and the system will return the output of the recommendation algorithm. This information will be displayed in a graph, where distance between titles is representative of the similarity between titles. When displayed, the titles will be displayed using their cover art, and the user can click on any title to see more information about that title.

If we have time before the end of the semester, we would like to expand our application and it’s user interface. First, we plan to add user profiles to the application. User information would be stored in a seperate database on our webserver and would be encrypted using PHP’s password and cryptography functions. Passwords would be stored in a salted and hashed database in the webserver.

We would store usernames, viewing history, and search history of users in our database. Additionally, we would allow users to search both by title/ID and by tag (using an advanced search GUI). This would allow the user to fine-tune their recommendations and searches. Additionally, in their user profile the stored data on their preferred tags would be visible and modifiable, and we would include a user self-test to allow the user to pre-tune the recommendation algorithm to some initial preferences.

In the users home screen the main features shown on the home-screen will be the recommended books, movies, and video games as well as the feature for users to input specific genres, titles, keywords, etc in order to recommend particular items. Other side features on the layout would include links to user profiles, friends list, saved for later list. Media-screen would show the title, picture, genres, etc. of said media item as well as having the “save for later” feature in order to save items onto their list to look into.

Additionally, we will explore options in our source API libraries to automatically update the database with new releases and fill in necessary data from the source API’s. Therefore, we keep the database current with new releases and new content for our users.

Further in the future, we plan to implement social networking functions into the app, possibly relying on Facebook’s API to gather friend and interest data. The friend and interest data would allow the recommendation algorithm to utilize collaborative filtering techniques to further refine recommendations. Additionally, we would set up the ability within our app to post user reviews and comments on media items. These user reviews and comments would be brought into the NLP functions of the application, and would contribute key words to the media listing weighted by user feedback to a particular comment or review.

**Features:**

***Core Features:***

* Microsoft Azure Cognitive Services Key Phrase Extractor to breakdown plot summaries and reviews into key phrases, which will then be stored as associated tags for titles.
* Search algorithm that, based on a list of input tags, returns media with shared tags. The search algorithm will order it’s results based on how many of the input tags are present in the returned media.
* Database with pre-gathered media titles, keywords, genres, plot summaries, and respective source database identifiers.
* Cache for images pulled from source API’s to populate the user interface.
* Conversion program to convert Goodreads, Google Books, IMDB, and RAWG API results and databases into a shared information format in our database.
* GUI frontend allows users to input a media title, then displays a graph with media titles grouped by strength of association (generated by search algorithm).

***Secondary Features:***

* User profiles
  + Users can create profiles in our service. These profiles will be stored in a webserver encrypted by PHP hashing functions.
  + User watch history (self-reported) and view history (automatically gathered) will be stored with their user profile.
  + The recommendation algorithm will save keywords from media the user has viewed. The frequency a particular keyword is encountered will be saved with the keyword, which will be used to determine user preference.
  + Users will also be able to self-report media preferences through a self-test GUI.
  + User preference information is stored with the user profile in the user database.
  + User will also be able to save a list of items they’re interested in to their profile.
  + The user will be able to report items they have seen or consumed elsewhere to the system, so the system knows what the user has seen and what not to recommend to them.
* General recommendation algorithm
  + The recommendation will be able to take user preference information in the form of an array of tags and output a recommended media item based on those tags.
  + Pick one for me, or “I’m Feeling Lucky”: asks the recommendation algorithm to pull a single ‘random’ recommended title based off of known user preferences and view history.
* User Ranking
  + Allow user ranking, use that to positively or negatively affect user association score with particular keywords and genres.
* User Interface
  + List of recommended genres and tag lists with associated recommended titles.
  + Profile user interface.
  + Media listing user interface.

***Future Features:***

* Social Networking
  + Utilize Facebook’s APIs to bring in friend connections and Facebook “likes” into the recommendation algorithm. Likes will be used to populate watched or consumed media, whereas friend connections will be used to perform collaborative filtering recommendations as well as tag association searches.
  + Allow users to comment and review media items, and incorporate those reviews and comments into the keyword extractor engine to populate keywords on media items. The prominence of these keywords will be influenced by the feedback of other users on reviews and comments.

**Technology:**

Platform

* Desktop application

Operating system

* Windows 10
* Mac 10.15.6

IDE

* Netbeans 8.2

Programming languages

* Java 11 through OpenJDK, using Hotspot JVM
* PHP 7.3.6
* SQL

3rd party libraries and tools

* JavaFX
* Goodreads API
* Google Books API
* RAWG API
* TMDB API
* Wikipedia Special:Export and Database Dump

Server software

* PHP MyAdmin 4.9.4
* mySQL 5.6.43

Communication software

* Discord
* Git
* Github
* Google Drive
* SMS

**Team Member Backgrounds:**

Peyton VanHook has experience working with Java, Javascript, C#, and C++, as well as Github, Git, Windows, and Discord. Additionally, they have worked with Google’s APIs before, and are somewhat familiar with working with online APIs and resources. They have experience designing and implementing user interfaces, working with databases (though not extensively in PHP or mySQL), and they have experience designing graphic assets for games and applications. In this project, they will be responsible for designing and implementing the recommendation algorithm (along with Sean Hopkins), building the initial database, and facilitating project organization and communication among team members. Additionally, they will be responsible for direct communication with the TMDB API.

Peyton White has experience working with Java, C#, SQL, and PHP, as well as working with and setting up web servers and server resources like databases. Additionally, he is familiar with Github, Mac OS, and Discord. He has experience implementing the front end of applications, and with setting up server communication with SQL and PHP. He will be responsible for setting up our web server and communication between the front end and back end, developing calls to API’s to keep the database current, and will assist in the development of the front end application. Additionally, he will be responsible for direct communication with the Goodreads and Google Books APIs.

Sean Hopkins has experience working with Java and C++, as well as Windows, Netbeans, and Discord. Additionally he has experience working with JavaFX, and designing and implementing program logic. He will be responsible for designing and implementing the recommendation algorithm (along with Peyton VanHook), working with the RAWG API, and implementing logic for the front-end functionality.

**Dependencies, limitations, and risks:**

*Dependencies:*

We will be importing data from a number of different sources, and will be reliant on a number of different API’s to keep that data current. We will have to design a common data format to convert each of our data sources into, and (for pulling plot summaries from Wikipedia and other sources) we will need to extract the data that we want with a loosely defined data structure. We will depend on a common JSON object to do this, which could be risky if the data is too incoherent to pull into a common database. Additionally, we need to make sure our server can handle the data that we’re putting in, so we will test our application comprehensively to make sure it is resilient to any communications errors.

Our front-end application will be almost entirely dependent on the back-end database in order to work. If there are communication issues with the database, then that could cause problems for how our user interface is populated and used. Additionally, if the database is not populated and the front-end requests data, we’ll need to have logic to handle that case. In these cases, we’ll work to overcome these problems by caching most recently viewed and used media items, and including code to handle error cases for if the database or cache are empty.

Our keyword extraction engine is dependent on Microsoft’s Azure Cognitive Services to work, however we may be extremely limited because of service pricing and budget. If we are unable to use Microsoft’s library, we will resort to one of a number of other libraries depending on which one can fill our needs. Most of these other libraries (especially the open-source or free ones) do not include keyword extraction, but include many of the basic functions of a NLP processor (tokenization, parsing, POS extraction, named entity recognition, classifiers, etc.). So we will need to design an algorithm of our own that uses those tools to pull out meaningful keywords for our search algorithm.

Additionally, we may run into latency issues or other communications issues with our client-server communication. We will tackle this problem by first focusing on making this communications infrastructure robust and capable of handling as many communication errors as it can.

*Limitations:*

Limited web server and JavaFX experience for all teammates is a definite limitation. In order to combat this we will reach out to those we know, in person and online, who have greater experience in these fields. We will also be looking to online documentation to inform our design process and to advance our knowledge in these areas. We will also be limited by how much we can make requests to APIs. Where this is relevant, we will limit our requests made to APIs and instead rely on pre-assembled databases or alternative sources of data to populate (example, Wikipedia) to source our summary and review information. The COVID 19 pandemic is a major limitation as well, especially if one of us contracts the disease. If this does happen, we will adjust our schedule and responsibilities accordingly.

*API Limitations:*

* For Microsoft’s Azure Cognitive Services, we will be limited to 5,000 transactions per month.
* For Goodread’s API, we will be limited to 1 request per second, but there is no limit on the number of requests we can make.
* There are no limitations on the Google Books, TMDB, or RAWG API’s, however the RAWG API terms of service requests users do not use their API to clone their database exactly.

*Risks:*

A risk that we all may come upon is limited web server experience. For example, we’re not yet entirely familiar with what issues we may face with client-server communication, such as latency issues or corrupted communications. We may have to adjust our plans for our database and webserver if we have major configuration issues. In general, we all have Discord and can communicate with one another to pool our knowledge together and work through topics we’re less familiar with. There are plentiful resources online that we can learn from to try to minimize the risk. COVID 19 will be another major risk. If one of us gets sick or if Austin Peay moves to fully online courses our efficiency and productivity as a group will be affected. Zoom and Discord have been very effective communications tools, but if one of us does contract COVID 19 then we will have to rescope our project and shift responsibilities and our project schedule, depending on the circumstances.

**Timeline:**

* September:
  + Week 1: (Sept 6-12)
* Set up github for all members and mess with JavaFX
* Set up server and database
* Look through API’s and get familiar with the JSON
  + Week 2: (Sept 13-19)
* Establish a basic GUI that connects with the database
  + Week 3: (Sept 20-26)
* Read in JSON for video game API
* Talk about algorithm and identify how to write it
  + Week 4: (Sept 27-Oct 3)
* Establish algorithm for video game recommendation
* Start looking into what to add to database
* October:
  + Week 1:
* Know what to add to database for video game part
* Work on GUI a little more
  + Week 2:
* Output a video game recommendation
* Work on getting that result to GUI
  + Week 3:
* Establish full GUI for video game recommendation and have output
  + Week 4:
* Begin looking at book and movie API and getting familiar with the JSON
* November:
  + Week 1:
* Read in JSON values for book and movies
* Find what to input into database
  + Week 2:
* Discover algorithm for movie and book recommender
* Output a result for both
  + Week 3:
* Finish full GUI and backend
* Double check everything front end and back end
* Talk about extra features
  + Week 4:
* Implement extra features
* Triple check everything
* Run final test