# GAME GENIE: THE ULTIMATE VIDEO GAME RECOMMENDATION SYSTEM

By

**Nicholas D'Amato, Bachelor's in Computer Science**

A thesis submitted to the Graduate Committee of

Ramapo College of New Jersey in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Summer, 2024

Committee Members:

Dr. Sourav Dutta, Advisor

Dr. Ali Al-Juboori, Reader

Dr. Lawrence D'Antonio, Reader

**COPYRIGHT**

# Dedication

To my family and friends who've supported me every step of the way

# Table of Contents

# List of Tables

# List of Figures

# Abstract

The video game market has a lot of variety and competition. It is hard for gamers to figure out a game to invest their time and money into. A lot of the leading video game companies such as Game Freak, Blizzard, EA, Bethesda, and Activision, created a highly valued reputation with good games in the past. However, in the present, they are making unpolished/unfinished games to get more money which people buy because of their high reputation. With an expert system that suggests games based on user ratings and personalized recommendations rather than popularity, gamers will have a better experience finding a game they prefer. Currently, there are a few of these video game recommendation systems either on the video game console or online, but all of them have flaws to them. These expert systems attempt to recommend games to people and include many different algorithms to predict and suggest games that match a specific user's preferences. However, these current recommendation systems lack personalized recommendations or lack in the data of games they have thus either giving the same recommendations to all users or giving the user a console-specific game. To solve this problem an expert system with personalized recommendations is needed. The system should incorporate a database that includes a variety of relevant games containing real-world data and put it into a web application. Then making sure the program worked properly and errors and bugs were eliminated from the web application. After completing and testing the proposed recommendation system, the final step was to observe the results and compare the system with other video game recommendation systems.

# Introduction

## 1.1 Background

One of the biggest forms of media that has been rising in popularity over the past few decades is video games. From Pong to Elden Ring, video games have changed drastically and so has the market. In the past, it was mostly Nintendo [20], Sega, and Atari making consoles and games, but nowadays anyone can make and sell a video game online. In present times, there are so many video games on so many consoles. You've got console-exclusive games, virtual reality games, mobile games, personal computer (PC) games, handheld games, and retro games, just to name a few not to mention the multitude of genres that the video games are categorized into. It can be overwhelming at times just how many games there are out there and finding the right device that can play them. These games can be very expensive as some games go for sixty or seventy dollars and games today can have budgets higher than Hollywood movies. Also, there is the fact that a lot of these games are a large time investment as some games can take over one hundred hours to beat. So, there comes the problem of a gamer trying to choose a game from the wide variety of video games that are available to them.

## 1.2 Problem Statement

Given that average gamers often encounter difficulties in selecting games they are likely to enjoy, further research is necessary to address this issue. Research should investigate the underlying reasons for this issue and explore why not all games are equally enjoyable to gamers. Recent findings indicate that many indie game developers have begun producing highly polished and well-rated games that, despite their quality, receive less recognition and fewer sales compared to 'AAA games.' These AAA games, characterized by their high budgets and extensive

promotion by large, well-known publishers, often command more attention and generate higher sales, even though they may sometimes lack the same level of polish or completeness. Average gamers often opt for popular games without conducting thorough research due to the overwhelming variety available. Conversely, games such as Pokémon Scarlet and Violet, despite their strong sales, received poor ratings from both gamers and critics due to issues related to lack of polish and the company's rush to release the game for financial gain. Therefore, the objective is to develop an expert recommendation system that provides personalized suggestions to video game consumers, enhancing the likelihood that they will enjoy and rate the games highly. Thus creating a technique-focusued/tool-focusued thesis which is improving upon what is already out there as well as creating an entirely new tool being a brand new video game recommendation system.

### 1.3 Significance of the Study

The computer science world is filled with tons of expert systems whether it be advice or making decisions in areas such as medical diagnosis like MYCIN and trading on the stock exchange like TrendSpider. There are also a lot of expert systems used for social media such as YouTube [30] or TikTok [28] to recommend content to users. These expert systems for social media use algorithms to recommend to users videos that the user would have a good probability of liking based on the other videos that the user likes. Most forms of media have some sort of expert system that recommends the user that media based on their likes and preferences. For movies and TV shows, there’s Netflix [19] and Hulu [13], for music, there’s Spotify [25] and Apple Music [2], and for social media, there’s Instagram [15] and TikTok [28]. But, there aren’t any well-known expert systems for recommending video games. There is Steam [26] which will suggest games to its users, but it’s only for PC games, not for anything found on consoles. Some

consoles will also have some sort of recommendation system, but again it’s only for games that are on that console. This works for people who only play on one console or only play on a PC, but most people either have a PC and a console or multiple consoles. However, many gamers own multiple devices, including both PCs and consoles. Consequently, there is no integrated system that provides game recommendations based on the user's entire array of gaming platforms. There are separate recommendation systems that aren’t built into the console or from Steam [26] that recommend games similar to something like GoodReads which is a separate recommendation system for books. As for games, this includes programs/websites such as Metacritic [17], IGDB [14], and RAWG [23] which contain user and critic ratings of games from all platforms. However, these systems do not offer the level of personalized recommendations that expert systems provide. Additionally, many of these recommendations are based on popularity rather than on detailed ratings, as seen with Metacritic [17]. Consumers face the challenge of selecting a game from a wide array of options and competition. Even when provided with suggestions, choosing a single game remains difficult, as it requires a significant investment of time and money without certainty of enjoyment. Many of these games represent substantial investments, further complicating the decision-making process for consumers. Nowadays, leading video game companies such as Game Freak [9], Blizzard [4], EA [6], Bethesda [3], and Activision [1], have taken over a majority of the video game market creating popular games with high reputations in the past, but the games coming out to be unpolished, and unfinished with an expert system that suggests games based on user ratings and personalized recommendations rather than popularity, indie companies will benefit and so will console games as the only other major expert system is for Steam PC games [26]. Typically, the average consumer opts for popular games, often referred to as 'AAA games.' These high-budget, high-profile titles are

produced and distributed by large, well-known publishers. Despite their high cost, these games are sometimes less polished or incomplete. However, indie games that have similar ratings and are cheaper aren't as popular but are just as good. Despite their quality, these games struggle to generate sufficient sales due to limited advertising and lesser-known reputations.

### 1.4 Objectives of the Study

This research work aims to achieve the following objectives:

- To obtain data of both games and users - This data should contain real-world data and enough data so that the algorithms can have as much data and information to work effectively as well as data of all games varying from all consoles.
- To find and implement personalized algorithms for the expert system that need to also work best for the information given and the data sets given and have a high probability of predicting a game that a user would like.
- To implement and visualize the expert system into a program - It will be implemented into a web application where not only will the results be shown using real data, but new users will be able to register to add their games and ratings to then see for themselves recommended games that will be predicted that they'd like based on personalized algorithms.
- To allow users to be able to conveniently track and mark down the games they've beaten and scored as well as games they are interested in, own, or consider to be one of their favorites.
- To allow users to find other games that are similar to their favorite games.

### 1.5 Structure of the Thesis

This thesis is organized as follows:

**Chapter 1** serves as the introductory chapter that introduces the topic at hand as well as the problem that needs to be solved and its significance and ends with the objectives that need to be done.

**Chapter 2** consists of the literature review that explains the previously existing techniques of recommendation systems, the algorithms used for recommendation systems, and other video game recommendation systems out there.

**Chapter 3** constitutes the research methodology chapter. It explains collecting the data for the system as well as implementing the algorithm for the data followed by how the web application would look.

**Chapter 4** presents the implementation of the proposed system. It discusses the design decisions and process of making the web application as well as implementing the algorithms into the web application.

**Chapter 5** presents and discusses the result and the challenges faced by the program such as bugs and errors and how they were fixed as well as testing the program out with real users and obtaining feedback.

**Chapter 6** describes and discusses how well the expert system turned out to be compared to other expert systems for recommending video games as well as ideas on how the systems could improve.

**Chapter 7** presents the conclusions and contributions of this research, besides the researcher's final thoughts.

# Literature Review

Before starting any large-scale project, one needs to do research and have a plan. The project started with a lot of research. What needed to be figured out was the algorithms that needed to be used and how they worked as well as how they would work with the data obtained. What also needed to be done was to look at other video game recommendation systems and how they worked as well as their strengths and weaknesses so that one could create a recommendation system better than what is already out there by taking all their strengths and avoiding their weaknesses.

## 2.1 Introduction to Recommendation Systems

The first thing that needed to be figured out was what a recommendation system is, how it works, and what makes a good recommendation system. In short, a recommendation system "suggests or recommends additional products to consumers" [29] based on data. These recommendation systems can be very personalized like Netflix [19] where each user will have different recommendations based on the user's data or inputs. Other recommendation systems such as the Nintendo eShop which uses popularity could be plain in which the recommendations could be the same for multiple users as the recommendations are based on other data such as the options data or some third-party data. But it's not just the data alone that is used to make recommendations for users, the data is put through numerous algorithms that vary depending on the data and they are used to narrow down or further personalize the recommendations for the user. These algorithms can be very complex making the recommendations as personalized as possible or very simple making the recommendations very plain where every user gets the same

recommendations. So for a good recommendation system, it should be simple, accurate, and have some form of personalized recommendations. This is a tough system to solve and get right as it heavily relies on the data. If there is not enough option data or user data, it's going to be hard for a recommendation system to produce accurate and personalized recommendations regardless of the algorithms. On the other hand, the algorithms are essential as well, even if the recommendation system has all the data it needs, the data needs to work well and fit with specific algorithms.

### 2.2 Algorithms for Recommendation Systems

To start the research, the expert system needed proper and personalized algorithms. Looking for algorithms that would work with a recommendation system and looked into what other recommendation systems used as algorithms. With help from Dr. Sourav Dutta, he ended up finding collaborative filtering [14], specifically user-based collaborative filtering which is where "...new items are recommended to a user based on the similarity measure with other users who highly rated the similar type of items." [14]. Following the article that Dr. Sourav Dutta was a part of (Top-k User-Based Collaborative Recommendation System Using MapReduce), for the recommendation system, the algorithm (User Similarity (Figure 1)) would pair the user with another user that has rated the same games and suggests the games they don't share which is ranked based on a formula that uses the other user's ratings as well as the user's average rating. The pseudo-code for user similarity would be something like this:

```
Algorithm 1: User Similarity
Input: user1 data, user database
Output: the list of all the data that user2 has that user1 doesn't,
sorted by formulas
```

```
1: for each user in the user database:
2:     user2 ← the user that has the most in common with user1
3: L ← the list of all the data that user2 has that user1 doesn't,
sorted by formulas
4: return L
```

(Figure 1: Algorithm 1: User Similarity)

The other algorithm that was looked upon and was interested in implementing was top-k suggestions (Figure 2). This would take the top-rated games, but it would be a bit more complicated as it would be from a filtered form of the data. This filtering would be based on user's preferences to personalize and narrow down suggestions of games. The pseudo-code for top-k would be something like this:

```
Algorithm 2: Top-k
Input: recommendation data and user preferences
Output: recommendations

1: recommendations ← recommendation data where recommendation data
contains user preferences and is sorted by ratings
2: return recommendations
```

(Figure 2: Algorithm 2: Top-k)

Lastly, the Naive Bayes algorithm [10] (Figure 3) was looked upon to further personalized recommendations by categorizing the users into specific groups and recommending games based on those groups. The reason for using this algorithm was for its basis in conditional probability. Specifically to find games with the highest probability that a user would engage with those games based on the conditions (the games and consoles the user profile contains) a user had. This is another form of filtering the data, but in this case, it is using individual data to categorize

and sort the individual data into a set group. A good example of this is used in emails as Naive Bayes [10] would categorize an email as spam or not spam depending on the words that are in the individual email where previous data is used as testing data to help determine and formulate what is considered spam email and what is considered a non spam email. In terms of the Naive Bayes [10] to be used with video games, the first idea was to take individual user data and recommend games based on genre which is based on the games the individual user has and thus categorizing users into groups based on what genre showed up the most and showing recommendations of that genre. That idea didn't work too well as games have multiple genres and in the case that there was a tie. So, instead, the groups were categorized by average rating. To do this, the idea was to use a training data set of users, enumerate the features (games and consoles the user owns), and pair it with that user's average rating. Then with new data, it would categorize a user and predict their average rating based on what the individual user has already, putting the user in a personalized average rating category. With the user in that category, we'd then recommend the user games that have that same average rating. The pseudo-code for Naive Bayes [10] would look something like this:

```
Algorithm 3: Naive Bayes
Input: user1 data and user test data
Output: predicted class

1: probability list of dictionaries ← user test data feature
appearance in each category where the category is the list of
dictionaries and the value is the probability
2: predicted class ← category with the highest added-up probability
based on user1's data
3: return predicted class
```

(Figure 3: Algorithm 3: Naive Bayes)

### 2.3 Existing Video Game Recommendation Systems

Looking at other recommendation systems that are out there, they all have their strengths and weaknesses. The most popular one is Steam [26] which is a video game market for PC games that will also recommend games. The recommendations use a similar algorithm of collaborative filtering [14] as it will show recommendations of games based on users like the individual user. Other recommendations that Steam [26] uses include recommendations that are similar games to the ones the individual user plays, what's popular, and sponsored games that pay Steam [23] to be placed on their front page. Based on similar games that the user likes is a pretty useful algorithm as it is personalized, but the algorithms based on popularity and sponsored games aren't as useful as those recommendations aren't personalized nor are they based on any of the data or input from the individual user. The program also only shows recommendations that are games only available on the platform. This is also relevant to other video game company stores such as the Nintendo eShop [20], the PlayStation store [21], and the Microsoft Store [18], as they show games only available to the platform. This is good if the user only has that one platform, but most of the time users own more than one platform for gaming. The problem with this is that these online video game stores will only show games that are on the platform and some games are on multiple platforms. So, sometimes users might see recommendations of games that they already have on other platforms. Thus some recommendation systems online solve this problem. Websites like Metacritic [17], RAWG [23], and IGDB [14] show all games on all consoles. Especially IGDB [14] which includes fan-made games and ROM hacks. For Metacritic [17] and IGDB [14], the games are rated by users and critics alike and are ranked and shown based on the rating of the game. Metacritic [17] takes it a step further allowing the user to filter the data by release year, platforms, and genre to narrow

down the recommendations more and make it more personalized as it is determined by the user's preferences. As for RAWG, instead of a normal number rating scale, it has a categorized rating scale of either exceptional, recommended, meh, or skip, and similarly to Metacritic [17], users can filter the rated games. These filters include tags (SinglePlayer, Multiplayer, etc.), platform, release date, and being able to order it by either popularity, date added, name, release date, and average rating, however, you can only select one per category, unlike Metacritic [17] where you can select multiple. Overall these websites are good as they have all games from all consoles, but they don't exactly recommend games as they just show the games with the highest ratings and thus have no personalized recommendations.

### 2.4 Gaps in Existing Research

After looking at the other recommendation systems out there for video games and seeing their strengths and weaknesses, what was found was what makes a good recommendation system and what was to become of a new recommendation system to improve the likes. Starting with the data, what was found was that one needed a good video game database that contains lots of games from every console and that each game has lots of information about it including details such as release date, platform it is on, genre, rating, and game mode (SinglePlayer, Multiplayer, etc.). As for the user data, what was known is that what was needed was at least information about what consoles the user owns and what games the user has already to eliminate recommendations of games that the user already has and to show only recommendations of games that are on consoles the user owns. As for the recommendations/algorithms, having the recommendations based on user ratings seems to be the most popular and best idea for getting the best results. Also having the user be able to filter the data by allowing the user to enter their preferences is a good idea to further personalize the recommendations. This is shown as video

game companies' online stores’ problem is that it shows only games available to that platform and shows recommendations that are paid for by other companies and as for recommendation system websites, they show every game but don’t have personalized recommendations. So, for the recommendation system, the goal was to make a system that solves all of these problems by showing every game and producing personalized recommendations.

With the algorithms decided and the recommendation system planned out, it was time to start the project. To do so, the plan was to find a valuable game database, make a simple user data set, code and implement the algorithms, and then design the web application to add everything in. Later on, what was added were some other features and data that were scraped from the web for real-world user data and ratings that were used to enhance the algorithms and add more depth to the project.

# Methodology

The process of making an expert system starts with finding the data. Scavenging the web, what was found was a good enough API to obtain a very strong game database that contained a lot of data. The other thing at hand was to scrape data from Metacritic [14] to obtain real-world user data which consisted of games, consoles, and ratings. Once the data was found there was also the fact of cleaning the data and preparing it to be a readable format for both a web application and for the algorithms. For the algorithms, what ended up being used were three different kinds: top-k, collaborative filtering, and Naive Bayes [11], each of which would personalize recommendations and rank them from most likely to least likely that the user would like the game recommended. Afterwards, the web application needed to be made being both the backend and the frontend. Using FastAPI [7] for the backend and React [24] for the frontend, the algorithms were put into the backend and the frontend took user input and displayed the results. Later on, user registration and other pages were made to make things more accessible and to allow new user profiles.

### 3.1 Research Design and Data Collection

To start, a large data set of games was needed. One that had a strong amount of information on each game. What was wanted was a data set that had at least the game name, what platform it was on, what genre(s) it was, what kind of game it was (Singleplayer, Multiplayer, etc.), the price of the game, and how long it took to beat the game. Looking through the web and searching through a bunch of different data sets from websites like Kaggle and such, most of the data sets either didn’t have enough data or didn’t have enough information on each game. So, after doing some more research what was landed upon was the API of IGDB [14]

which is one of the online websites rating video games, but also one of the largest video game databases. After obtaining the API code, one was able to access the IGDB database through Python. At first, it was hard to know how to use it, but after looking through the endpoints, extracting the data was easy and obtaining exactly what was wanted. What ended up getting extracted was the game's ID, name, genres, platforms, similar games, rating, game modes, and rating count which were all endpoints for the game path. Which for the most part was everything that was wanted except for how long to beat the game and the price of the game. Extracting the data was a bit tricky as the data set contained over 250,000 games, but the request only allowed 500 games at a time, so by putting it into a for loop and changing the ID numbers obtained by 500 every loop, the data was able to be obtained. Also, the data was formatted into a JSON string which ended up being converted into a pandas dataframe. A little more research was done to see if the specific data that was missing could be obtained which were the price and how long to beat the game, but most of the data sets found were small compared to the 250,000 games that were from IGDB [14]. Where some data sets only had around 5,000 matching values, so what ended up happening was that that data was not included. As for the data obtained, the IGDB data frame was created and had columns that were filled with data that were in the format of lists of dictionaries containing ID numbers for each name as the key and the information as the value. Each value was extracted and was made into a list of those values instead as the key ID number was not needed. Also, to clean it up, the data was put in game ID order as well as removed all missing values and games that had a rating of 0. With that, what was ended up with was a clean data set of 22,476 games as a lot of them either had missing values or were shovelware games (low-budget, poor-quality video games, released in the hopes of being purchased by unsuspecting customers) (Table 1).

| | id | game_modes | genres | involved_companies | name | platforms | rating | release_dates | similar_ga |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | Singleplayer | Shooter, Simulator, Adventure | EidosInteractive, LookingGlassStudios, SquareEnix | ThiefII:TheMetalAge | PC(MicrosoftWindows) | 86.298807 | Mar21,2000, 2000, May22,2012 | Thief:TheDarkPr Thief:DeadlyShac |
| **1** | 2 | Singleplayer | Simulator, Adventure | LookingGlassStudios, EidosInteractive, SquareEnix | Thief:TheDarkProject | PC(MicrosoftWindows) | 86.579238 | Nov30,1998, 1998 | ThiefII:TheMeta Thief:DeadlyShac T |
| **2** | 3 | Singleplayer | Shooter, Simulator, Adventure | IonStorm, EidosInteractive, SquareEnix | Thief:DeadlyShadows | PC(MicrosoftWindows) | 82.006129 | May25,2004, Jun11,2004, Mar29,2007, May25,2004... | ThiefII:TheMeta Thief:TheDarkPr |
| **3** | 4 | Singleplayer | Shooter, Adventure | EidosMontréal, SquareEnix | Thief | PC(MicrosoftWindows) | 70.253640 | Feb25,2014, Feb25,2014, Feb25,2014, Feb25,2014... | ThiefII:TheMeta Dishon RogueWar |
| **4** | 5 | Singleplayer | Role-playing(RPG) | BioWare, BlackIsleStudios, InterplayEntertainm... | Baldur'sGate | Linux | 86.002845 | Oct14,2015, Dec21,1998, Nov30,2012, Oct14,2015 | Borderla PokemonS OrientalBlue:Ao |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **22471** | 269135 | Splitscreen | Shooter, Adventure | RemedyEntertainment, EpicGames, NitroGames, D3... | AlanWake:TheSignalRemastered | PC(MicrosoftWindows) | 57.248435 | Oct05,2021, Oct05,2021, Oct05,2021, Oct05,2021... | Don'tKnockT TheDarkO Immortal:Un |
| **22472** | 269155 | Singleplayer | Shooter, Adventure | RemedyEntertainment, EpicGames | AlanWake:TheWriterRemastered | PC(MicrosoftWindows) | 60.131088 | Oct05,2021, Oct05,2021, Oct05,2021, Oct05,2021... | Masoc Don'tKnockT HouseofCarava |
| **22473** | 269849 | Singleplayer | Simulator, Indie | ReflectStudios | DeadSignal | PC(MicrosoftWindows) | 78.542773 | Oct20,2023 | Masoc AnotherBrickinthe SupremeF |
| **22474** | 272249 | Singleplayer | Puzzle, Indie | FLEB | 20SmallMazes | PC(MicrosoftWindows) | 71.797753 | Feb16,2024 | RustyLake RustyLake:R Don'tKnock |
| **22475** | 274920 | Singleplayer | Simulator, Strategy, Indie | NoktaGames | SupermarketSimulator | PC(MicrosoftWindows) | 60.424163 | Feb20,2024 | UnclaimedV Project AnotherBrickin |

22476 rows × 9 columns

(Table 1: clean game dataset)

After the research of finding the game data set, making the user data set was the next step. At first, it was simple, the data set contained the columns: userID, username, password (for the web application), backlog (games that the user wants to play/own), beaten_games (games the user has beaten), fav_games (the user's favorite games), consoles (consoles the user owns), and ratings (ratings the user gave for each game they beat). The database was filled with just a few example users that were made up first to test how it would work with the algorithm and the game database (Table 2).

| | userID | username | password | backlog | beaten_games | fav_games | consoles | ratings |
|---|---|---|---|---|---|---|---|---|
| 0 | 55 | ndamato | password | DavetheDiver, Balatro | AHighlandSong, LiesofP | XenobladeChronicles, CrossCode | Switch, PS4, PC | {'XenobladeChronicles': 95, 'CrossCode': 92} |
| 1 | 3456 | User1 | password | Persona4, DragonQuestXI:EchoesofanElusiveAge, ... | Persona5, FinalFantasyX | FinalFantasyIX | Switch, PS4, PS1, PS2 | {'FinalFantasyIX': 89} |
| 2 | 3495 | User2 | password | ChronoTrigger, Persona5, SeaofStars, ChainedEc... | EarthBound, FinalFantasyVIIRebirth, Mother3 | PaperMario:TheThousand-YearDoor, XenobladeChro... | Switch, PS4, PS1, PS2, GBA, GameCube, SNES, PC | {'PaperMario:TheThousand-YearDoor': 85, 'Xenob... |

(Table 2: basic user data set)

### 3.2 Algorithm Development

The first algorithm that was used, was one that Dr. Sourav Dutta used in his project which is collaborative filtering. He used the algorithm to predict movie titles that users would like which works well for the data set of games since it was a similar data set. However, his code was using MapReduce while the code for the game data set was using Python. So, what ended up happening was the translation of the code to Python and making the functions: genAverageRating, genUserSimilarity, and genRecommendation (The ones Dr. Sourav Dutta, used). What genAverageRating (Figure 4) did was take all of the users' ratings add them up and divide them by the number of ratings to get every user's average rating which ended up stored in the data frame adding a new column called "average_rating" with pseudo code that looked like this:

```
Algorithm 4: genAverageRating
Input: user database
Output: none, just adding a new column being the average rating to
the user database for each user

1: for user in user database:
2:     for rating in ratings:
3:         sum ← sum + rating
```

```
4:          number of ratings ← number of ratings + 1
5:      user's average rating column ← sum/number of ratings
```

(Figure 4: Algorithm 4: genAverageRating)

The genUserSimilarity function (Figure 5) took the specific user and compared that user to all other users to find the one that has the most similar taste which was to find the user with shared beaten games, as well as using a formula to find the user with the highest similarity score with pseudo-code for genUserSimiliairty looking something like this:

```
Algorithm 5: genUserSimilarity
Input: user1 and user database
Output: max similarity score and index of user with the highest
similarity score

1: max similarity score ← 0
2:      for user in user database:
3:          L ← beaten games user and user1 have in common
4:          sn ← 0
5:          sdi ← 0
6:          sdj ← 0
7:          similarity ← 0
8:          for each l in L:
9:              sn ← sn + (rating of l from user1) * (rating of l from
user)
10:             sdi ← sdi + (rating of l from user1)^2
11:             sdj ← sdj + (rating of l from user)^2
12:             similarity ← sn/(sqrt(sdi) * sqrt(sdj))
13:             if similarity > max similarity score:
14:                 max similarity score ← similarity
15:                 index for user with the highest similarity score ←
user's index
16: return max similarity score and index for user with the highest
similarity score
```

(Figure 5: Algorithm 5: genUserSimilarity)

Then with the genRecommendation function (Figure 6), the specific user and the best pair for that user are taken from the genUserSimilairty function, and by using a specific formula that uses the user's average rating, the beaten games that the paired user beat in which the specific user hasn't beat are recommended to the specific user and are ranked based on the paired user's ratings of those games with pseudo-code for genRecomendation looking something like this:

```
Algorithm 6: genRecomendation
Input: user1, index for user with highest similarity score, max
similarity score, the average rating of user1, and user database
Output: recommendations

1: recommendations <- {}
2: sn ← 0
3: sd ← 0
4: user2 <- user whose index is that of the index for user with
highest similarity score
5: for each beaten game that user2 has that user1 doesn't have:
6:     rating ← rating of beaten game from user2
7:     sn ← sn + (rating - average rating of user1) * max similarity
score
8:     sd ← sd + max similarity score
9:     recommendations[beaten game] ← average rating of user1 + sn/sd
10: recommendations ← recommendations sorted by value
11: return recommendations
```

(Figure 6: Algorithm 6: genRecomendation)

After testing and tuning to make sure the user-based collaborative filtering [14] worked well, it was time to work on another algorithm which ended up being top-k. This algorithm would take user input of their preferences and based on those preferences the algorithm would return games of those preferences and be ranked based on the ratings found in the games data frame. This ended up being the function "filter_dataframe" (Figure 7). This function asked for what genre(s)

and game mode(s) the user liked as well as what consoles the user owned and based on those answers, the function would output games that contained the specific genre(s) the user chose, the specific game mode(s) the user chose, and the games that were only on the console(s) that the user owns that the user imputed and as said before the games are ranked based on the ratings that the game data set has where the function of filter_dataframe had pseudo-code like this:

```
Algorithm 7: filter_dataframe
Input: game database, genres, game modes, platforms, number of
suggestions, and beaten games from user1
Output: filtered game database

1: filtered game database ← game database only containing genres,
game modes, and platforms as well as excluding all games from beaten
games from user1 and is sorted by rating and only showing the top
results based on the number of suggestions
2: return filtered game database
```

(Figure 7: Algorithm 7: filter_dataframe)

### 3.3 System Architecture

With these two algorithms made and implemented into the data sets, it was time to work on the web application. At first, using Flask [8] was the first idea as it was a popular Python backend, so it would be easy to bring over the algorithms since they were also in Python. But in the end, FastAPI [7] seemed like a better option as it worked better with large amounts of data and it also used Python. Learning FastAPI [7] was a bit tough as it is fairly new, so learning the etiquette of the language was a bit of a learning curve, but after a bit of practice, the next step was to add the code of the algorithms into it. Once that was done, the data sets had to be implemented into FastAPI [7] as well. To do this PostgreSQL [22] was used and a bit of Python code was used to store the two data sets so that the FastAPI [7] backend can access it. However,

the FastAPI [6] code receives the datasets as JSON strings which are then converted back into a pandas dataframe. With this, the next step was to test the backend to see if it outputted the correct information and once it did, it was time to move onto the frontend. With the frontend of the code, React was used which used JavaScript and CSS.

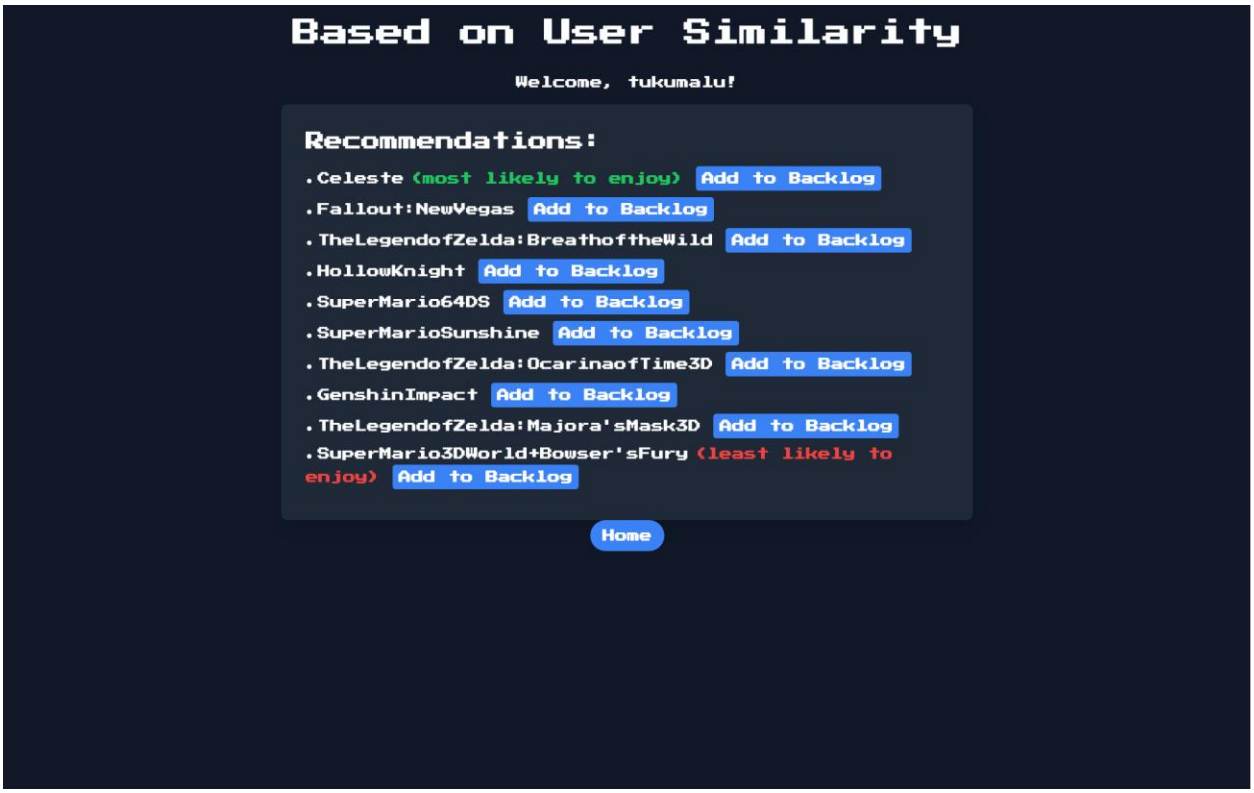
# Implementation

### 4.1 Development Environment

At first, a simple home page was made (Figure 8) which had two buttons: the collaborative filtering button and the top-k suggestions button which would redirect to a different page based on the button that was clicked.


(Figure 8: home page)

For the collaborative filtering page (Figure 9), the user would input a username and the page would suggest games based on the user-based collaborative filtering algorithm.



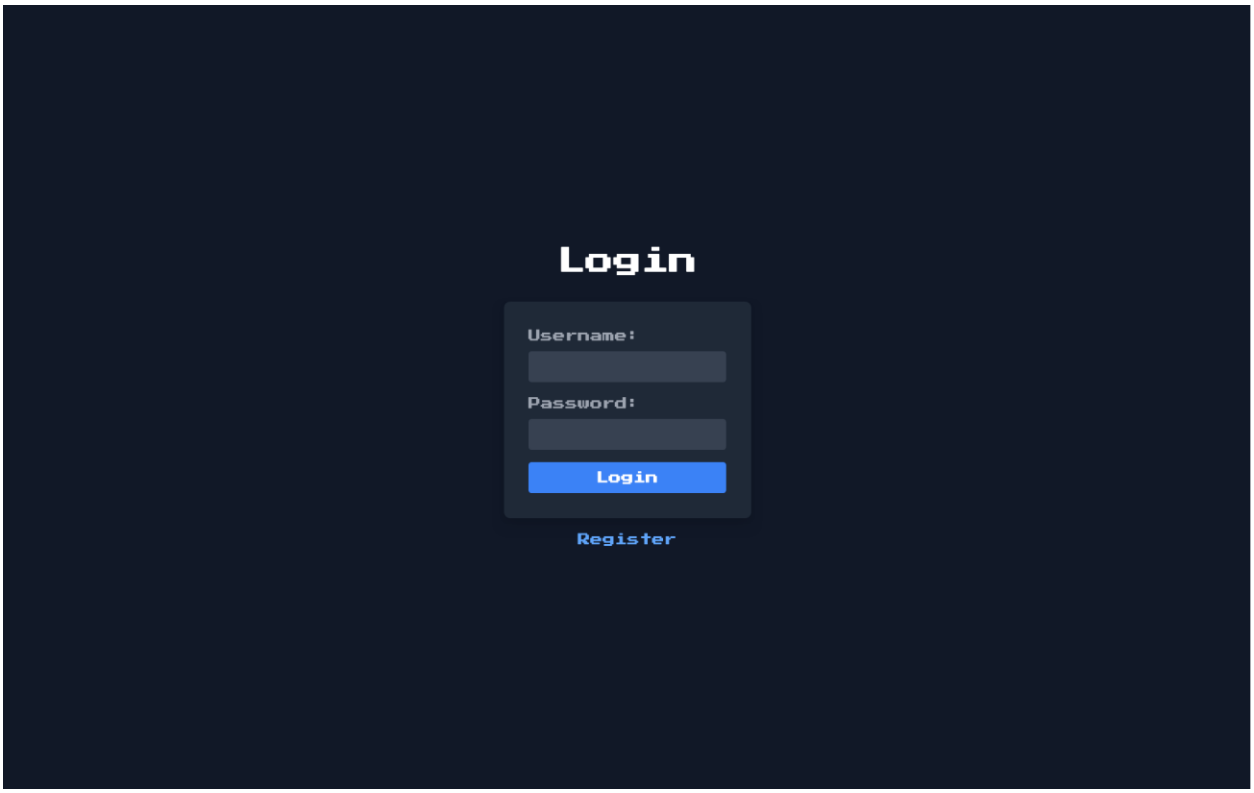

(Figure 9: collaborative filtering page)

As for the top-k suggestion page (Figure 10), the user would click on the checkmark boxes of the genre(s) and game mode(s) as well as the console(s) they owned as well as adding an input box for the user to enter how many suggestions they'd like.


(Figure 10: top-k suggestions page)

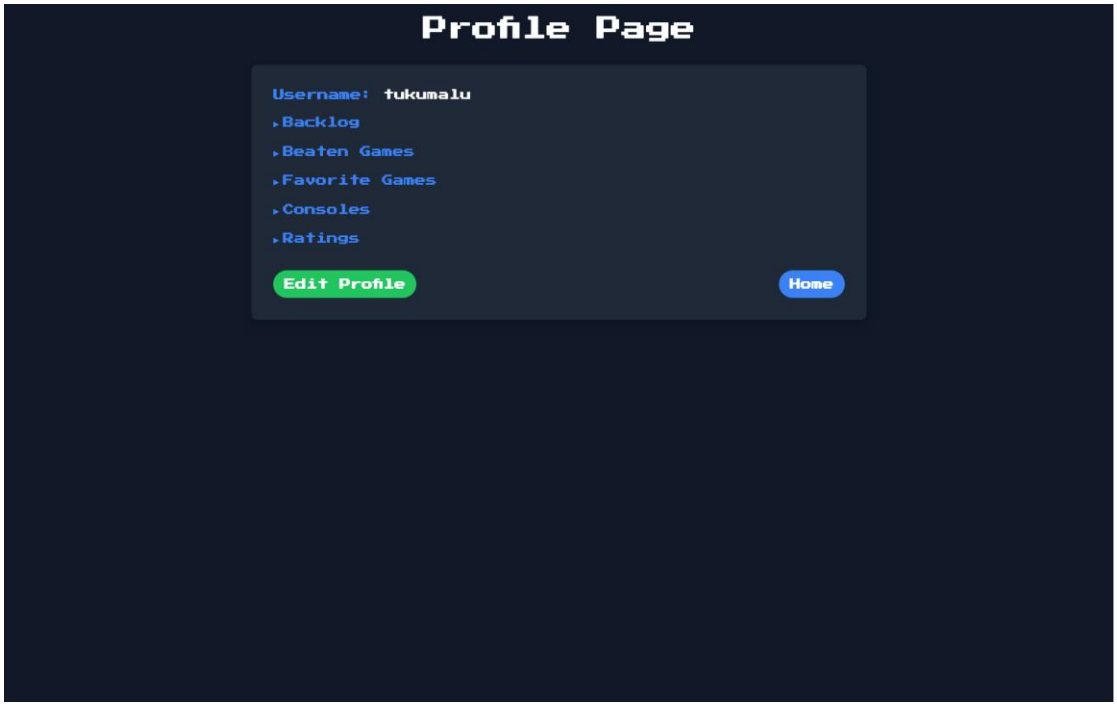
### 4.2 User Interface Design

Once the home page and algorithm pages were done and working, the next step was to make a user login page (Figure 11) which would be the first page and would only move onto the home page if the user entered a username and matching password that was in the user data frame.

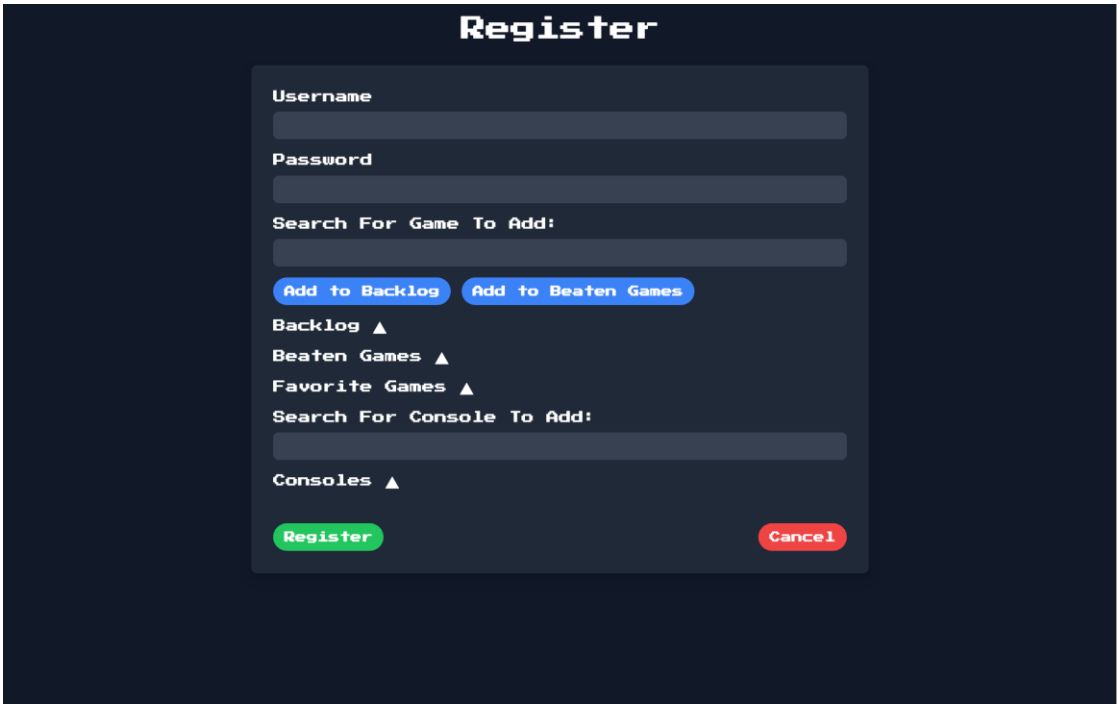



(Figure 11: login page)

With this data, the decision to remove the input box for the collaborative filtering page was made and was replaced with the data based on the username that was entered for the login page. After that, a simple profile page was made (Figure 12) that showed the user's data and could be accessed through the home page as well as a register page for new users to register(Figure 13).




(Figure 12: profile page)




(Figure 13: register page)

Now the register page would be for new users as it would ask for a username, password, and games and consoles to add. The games to add section would have an input box that asked for a game and then the user could select whether to add that game to their backlog or their beaten games list if it was a valid game found in the game database. If the user added it to their beaten games list, the program would ask the user how they would rate the game and if it were one of their favorite games or not, and if it was, the game would also be added to the list of their favorite games. As for the console input box, the user would enter the console(s) they had individually and if it was a valid console from the game data frame, it would be added to the user's console list. This valid console list was made by finding all the unique values from the platform column in the game data frame. Similarly, the edit profile page was made (Figure 14) which could be accessed through the profile page where users can add or remove games from their beaten games, backlog, or favorite games list.


(Figure 14: edit profile page)

Afterward, another page was added called the “SimilarGamesPage” (Figure 15), this page would take the user’s favorite games and upon clicking on one of them the page would show games that are similar to that game which is stored as one of the columns in the games database.


(Figure 15: similar games page)

Once that was done navigation buttons were added like a home button, logout button, edit profile button, and registration button so that the user can navigate between the pages. What was also added were other buttons such as add to backlog which would show next to the suggested games for users to keep track of and add games that interested them based on the suggestions. The button would also appear as “remove from backlog” if the game suggested was already in their backlog. As well as adding a learn more link to each game to increase the simplicity for the user as clicking the link will direct the user to a Google search of the game with the added word

“video game” so that it shows the information about that game for the user if they’re interested. The checkboxes for the top-k suggested page have also changed the checkboxes as only the console checkboxes the user owns appear. Also for simplicity, an autocomplete window was added for both the search games and search consoles input box so users can see which games and consoles are valid as well as not having to type the full thing but they can click on the full title instead. And for the top-k suggestions and similar games page, if the game was already in the user’s beaten games list, the change was made that that game won’t show as a suggested game as it is a game they already beat just like how the user similarity page won’t show a game the user already beat. As for if they are already in their backlog, the game will show, but the button next to it will appear as removed from the backlog. As for the edit profile page, things were made a bit more simple for adding games to favorites and removing games as the remove button was turned into a trash can and the add to favorites button was changed to be an outline of a star and if the game is in the list of their favorite games, the star would be yellow. Also for the backlog list, a checkmark button was added which is for users to click when they beat the game as the program will ask for a rating and if it is one of their favorite games as well as removing it from the backlog list and adding it to the beaten games list as well as adding it to the rating list and the favorite games list if the user clicked ok for if it was their favorite game. A few bugs were found which were shortly fixed. These included fixing it so that users can’t have a game both in their beaten games and backlog list, fixing the issue that if a user removes a game from their beaten games list for whatever reason, it would get rid of the rating as well, as well as only showing suggested games that the user can play (games that can be played based on the consoles the user owns). Then some formatting and changes to the style of how the web application looked were made. Using Tailwind [27] and other CSS libraries, the application now had more of a video

game look to the web pages and everything was made sure to be working and looked right. Once that was all done what was left was to set out to find real-world user data to test the algorithms on real data.

### 4.3 Backend Infrastructure

To obtain the real-world data, Metacritic [17] seemed like a good place to find individual user rating data as IGDB [14] didn't have that data. Going through Metacritic [17] the first idea was to try and obtain the data by hand by taking random users and using the register page of the web application and marking down the username, beaten games (games the user rated), ratings for each game, and the consoles each user owned. This worked at first, but some of the users had over 100 reviews which would've been very time-consuming to mark down all those reviews. So, instead, the other way was to scrape the data from the web. To do this, the tool Chrome WebDriver [5] was used which took the exact URL of the user page and by searching through the CSS selectors the program would find and scrap the necessary info of the user. With that, the data would then be added to the user database which would now be filled with real users as the fake example users were removed. However, it wasn't that easy. The main problem arose which was the fact that the Metacritic [17] data wasn't the same as the game data set. Things like console names were named differently and were all in caps as well as game names with accent marks such as Pokémon didn't have those accent marks and the fact that Metacritic [17] has a scoring scale of 0-10 while IGDB [14] has a scoring scale of 0-100. So, to solve these problems the game database was edited and corrected by removing the game tile's accent marks, matching the Metacritic [14] console names from the real-world data to the game data set console names, and multiplying the rating score by ten to match that of IGDB ratings. With that, there was still one other problem that needed to be solved which was that the program wasn't getting all the

games and reviews. This was because if the user had a lot of reviews, you needed to scroll to the bottom to load all the reviews which was manageable with a function that made the webpage scroll to the bottom to load everything. The last thing that was done was to make sure the game recorded was in the game data set as not all the games on Metacritic [17] were in the game's data set. With all of that 100 real-world users were obtained and to clean it all up duplicates of usernames were removed as well as for each user if they had duplicate game reviews, they were also removed (Table 3).

| | userID | username | password | backlog | beaten_games | fav_games | consoles | |
|---|---|---|---|---|---|---|---|---|
| 0 | 30ba1c71-7cfd-4168-bf0b-79d645ce3300 | tukumalu | password | [] | ['TheLegendofZelda:TwilightPrincess', 'Neighbo... | [] | ['NintendoGameCube', 'PlayStation3', 'PlayStat... | {'NieRRe |
| 1 | ad4674ec-ebe4-46ec-8d87-d1c69b2a092d | Bagubuns | password | [] | ['HollowKnight', "TheLegendofZelda:Majora'sMas... | [] | ['NintendoGameCube', 'Nintendo64', 'Nintendo3D... | {'Celes |
| 2 | 76c07864-6e85-4430-8531-06f47aa9d962 | Kino1337 | password | [] | ['Castlevania:SymphonyoftheNight', 'MetalGearS... | [] | ['PlayStation3', 'PlayStation5', 'Nintendo64',... | {' |
| 3 | 3b235579-eebf-496b-9e91-91bb7bca03d7 | igorbarazzetti | password | [] | ['TheLegendofZelda:TearsoftheKingdom', 'Prince... | [] | ['Nintendo64', 'NintendoSwitch', 'PC(Microsoft... | {'Under |
| 4 | eadf3390-fdbb-4f3c-b703-4b964720da7c | ElderMist | password | [] | ['DiscoElysium:TheFinalCut', 'PrinceofPersia:T... | [] | ['PlayStation5', 'Nintendo64', 'XboxOne', 'Unk... | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 95 | 2063553f-5c7d-4885-aab5-4fcf96286eb6 | lazatoy | password | [] | ['SimCitySocieties', 'GrandTheftAutoV', 'Flapp... | [] | ['PlayStation3', 'Nintendo64', 'Nintendo3DS', ... | {' |
| 96 | 6d9e1dcf-979b-40ac-b5fa-f3adeea7031c | joaopontesvaz | password | [] | ['GranTurismo6', 'MaxPayne3', 'AgeofEmpiresIII... | [] | ['NintendoGameCube', 'PlayStation3', 'WiiU', '... | |
| 97 | d815121a-5560-4144-8e80-e283c4257d19 | artoldtonelic | password | [] | ['PrinceofPersia:TheSandsofTime', 'MetalGearSo... | [] | ['Nintendo64', 'PlayStation', 'PC(MicrosoftWin... | |
| 98 | a492c406-142b-4aa4-82a8-eca20b2a3efd | XRAIN | password | [] | ['Batman:ArkhamCity', 'Singularity', 'Vanquish... | [] | ['NintendoGameCube', 'PlayStation3', 'PlayStat... | |
| 99 | 5a19a5e1-6ac4-4d78-89cf-45c7aea317c8 | bestbloodyday | password | [] | ['BatenKaitosOrigins', 'TheLegendofZelda:Twili... | [] | ['NintendoGameCube', 'PlayStation2', 'Nintendo... | 'Neec |

100 rows × 9 columns

(Table 3: 100 real users data set)

### 4.4 Algorithm Integration

With the real-world user data, the user database was updated to be that of the 100 real-world users, it was tested with the web application and was made sure that everything was working properly. Once that was done, with the help of Dr. Sourav Dutta, another algorithm was decided to be added which was the Naive Bayes algorithm. The first step for this algorithm was to encode all categorical data into numbers. In Python, each unique feature the user had (games and consoles), was encoded by giving it a specific number which was made into a function called create_feature_vectors (Figure 16) which had pseudo code like this:

```
Algorithm 8: create_feature_vectors
Input: user database
Output: feature vectors and targets

1: unique games ← insert all unique games from user database
2: unique consoles ← insert all unique consoles from user database
3: feature vectors ← encode each unique game and console with a
different number
4: targets ← all average rating appearances
5: return feature vectors and targets
```

(Figure 16: Algorithm 8: create_feature_vectors)

Once every user's data (the real-world user data scraped from the web) was encoded, it was time to split the data into test and train sets with the test set being twenty percent and the train set being eighty percent. Once that was done, using the training data, the prior probabilities were calculated which meant the probability of the average rating data based on the test set a function was made called calculate_prior_probabilites (Figure 17) which had pseudo code like this:

```
Algorithm 9: calculate_prior_probalities
Input: targets
Output: prior probabilities
```

```
1: average rating, count ← all unique average ratings with their
respective count appearance
2: total ← size of targets
3: prior probabilities ← average rating: count/total
4: return prior probabilities
```

(Figure 17: Algorithm 9: calculate_prior_probabilities)

Prior probabilities was a dictionary of values where the key was the unique average rating category and the values were the probability of that average rating category which was calculated by the number of users that had that average rating divided by the number of users which was 80 since it was the training data. Once that was calculated, the next step was to calculate the likelihood of the training data, this was done by counting the number of times a unique feature (game or console) appeared in each average rating category plus one divided by the number of users in that average rating category plus the number of unique average rating categories which was made into a function called cacluclate_likelihoods (Figure 18) which had pseudo code like this:

```
Algorithm 10: calculate_likelihoods
Input: features and targets
Output: likelihoods

1: likelihoods ← {}
2: classes ← unique values in targets
3: for class in classes:
4:     total_count ← number of instances class appears in targets
5:     for feature in class:
6:         feature count ← number of times feature appears in class
7:     for value in feature count:
8:         likelihoods ← (feature count[value] +1) / (total_count +
size of classes)
```

```
9: return likelihoods
```

(Figure 18: Algorithm 10: calculate_likelihoods)

The reason for the plus one for likelihoods is for Laplace smoothing which is to make sure the probability is never zero. With this information, the average rating for new users was able to be predicted. To do this the first step is to go through every average rating category and within that go through every feature this new user has. If the feature the new user has is in the average rating category, the probability is calculated by multiplying the probability of that average rating category (prior probability) by pi and then by the probability of that feature in that average rating category (likelihood). The probabilities are then compared to find the highest probability in which the user's predicted average rating will be the average rating category with the highest probability which was made into a function called predict (Figure 19) with a pseudo-code like this:

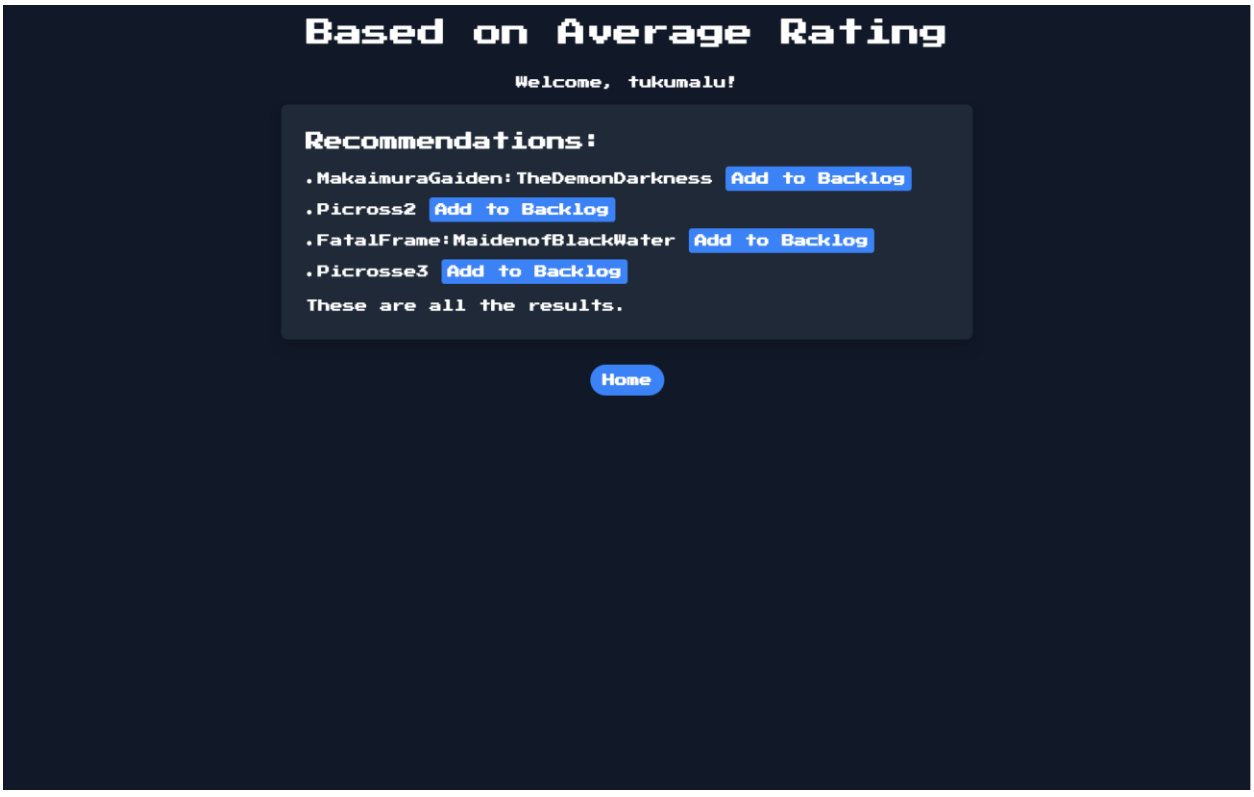
```
Algorithm 11: predict
Input: user1's features, priors, and likelihoods
Output: predicted class

1: user probability ← -1
2: max probability ← -1
3: predicted class ← none
4:     for class in priors:
5:         probability ← priors[class
6:          for feature in user1's features
7:              if feature is in likelihoods[class]:
8:                  user probability ← probability * pi *
likelihoods[class][feature]
9:              if user probability > max probability:
10:                 max probability ← user probability
11:                 predicted class ← class
```

```
12: return predicted class
```

(Figure 19: Algorithm 11: predict)

Once that was done, the mean squared error was tested which was roughly around 13.67% which was a bit high. So, after testing some things out, what was found was that rounding up the average ratings of the data to the nearest whole number drastically lowered the mean squared error to 6.03%. After getting the ability to predict a user's average rating, it was now time to apply that to the web application. With the predicted average rating of a new user, using the game dataset, the web application would recommend games that had the same rating as the predicted average rating and be sorted by the number of rating scores (popularity) and would be displayed as the Naive Bayes page (Figure 20).




(Figure 20: Naive Bayes page)

The web application would also ask the user for the number of suggestions (top-k) before showing the results similar to the top-k algorithm used to recommend games based on preference.

Once that was done, the project objectives were complete. With two real-world data sets (game data set and user data set), three algorithms (collaborative filtering [14], Naive Bayes [11], and top-k), and a user registration, profile, and navigation system (web application) the project had a lot of data and moving parts to it. The next step to do was to test the application for bugs and errors and to fix any of those bugs and errors as well as have real users test the application and see if it was doing its job correctly and accurately.

# Results

After obtaining the data, making and implementing algorithms for that data, and putting it out into a web application, the project was complete in terms of functionality. What was left was to test it out, see and figure out if the project had any bugs or errors, and see how well the project works. Testing out every web page, and every algorithm, and testing it with the real user data obtained.

### 5.1 Challenges and Solutions

Starting with the login page, it was tested to see if all users could log in and only log in with usernames that were in the user data set (using the real-world user data set) with the matching password for that username. To do so, it was checked to see if a username using a different user's password would work, but luckily it didn't. Then moving on to the register page, it was checked to see if a user could register with the same username as one in the user data set which it could at first, but after making a change of not allowing a new user to put a username that was already taken by giving the user an error message when the user clicked on register allowing the user to not register if the username is already taken. As for the password part, the issue where the user could register without a password by having the program put the typing cursor to the password section if the password was blank was fixed by not allowing the user to register without a password. Moving on to the adding games section, the problem of if a game was in the beaten games list couldn't appear in the backlog list and vice versa was fixed. Also the problem of if a game was taken away from the beaten games list, it would get rid of it and its rating in the rating list as well as get rid of it in the favorite games list if it was in the favorite games list was fixed. Not having a game that can be in the backlog and the beaten games list

prevented the problem of a game from being in the favorite games list and the backlog list. As for the console input and list, the autocomplete window was made simpler where the game search would autofill the title and then the user would click on whether to add the game to the backlog or beaten games list, for the console, since there wasn't an option when the user clicked on the name the program would then just automatically add that console to the list instead of filling in the input section and then clicking add. Also for simplicity, all the lists were made to be drop-down menus that appear closed at first to make things less cluttered especially if the user has a lot of games and consoles to add to their profile.

Moving on to the home page, all the navigation was checked to see if it worked which it did so there weren't any problems there. Then for the collaborative filtering page, if the user didn't have any or enough games rated in which case it wasn't matched with a user, the program would tell the user to rate more games. Also, since each game has a score based on the formula for the suggestions, a little note to the side of the top game was added saying the user would most likely enjoy this game and the game at the bottom saying the user would be least likely to enjoy this game. As mentioned before each game had the button add to backlog, and when the user clicked on the button it would say remove from the backlog. This worked, but when the user would add to the backlog and then come back to the page, it would say add to backlog again which created the problem of adding the same game to the backlog. This was also the case for the other pages and to solve this problem check was made in the code where when the user clicked on the page the program would first check to see if the game was in the user's backlog and if it was the button would then first appear as remove from backlog instead of add to the backlog. Then for the based on preference page, there was a problem where games that the user had beaten would show up. This wasn't relevant for the collaborative filtering page since it

would only show games the user hasn't beaten by showering only games that the paired user has beaten that the other user hasn't. So to solve this problem game suggestions that the user has already beaten were removed as well as if they had it in their backlog as well. This problem would also be solved for the other pages as it had a similar problem. Other than that, the based on preference page worked well at filtering the data frame and showing the number of suggestions. Then based on games similar to the user's favorite games, at first all the games were shown, but that didn't work well or efficiently as the user should be able to see which games were similar to which specific favorite game. To do so cleanly and efficiently, a list of favorite games was made, and when a user clicked on one, the user would be presented with similar games to that specific favorite game. Then for the Naive Bayes page, the only problem was that if the user entered a higher number of suggestions than the number of suggestions only available, the program would only show the maximum amount it has. So to fix this problem, a message at the bottom of the page saying this is all the suggestions available was also added to the preference page even though that error wasn't as common for that page. Then for the profile page, the only thing that changed was making the lists drop-down menus to make things look cleaner especially if the user had a long list of games and consoles. As for the edit profile page, one problem was that some of the game titles were a bit long and moved some of the buttons which was fixed by cutting off the title to "..." after 44 characters. Since some games had these long titles, the buttons were also made simpler. Instead of having a button that said beat it for a game in the backlog to add it to the beaten games list, a checkmark button was made, and as for the remove button, a trash can button replaced it, and lastly, to add a game to favorites, a star button was made and would first appear as an outlined star, but when clicked on would add the beaten game to the favorite games list as well as appear as a filled out yellow star. The edit

profile page would also have the problems that the register page had solved as well such as a game being in the backlog and beaten games list or a game being in the favorite games list and not in the beaten games list.

### 5.2 Comparative Analysis

With those changes, the web application was much more efficient, cleaner, and more accessible as well as not having any problems for users. So, then it was time to observe the results that the web application had to offer. First checking the user similarity page, the algorithm seemed to be working, the user was being matched with another user that had some matched, but not all matched beaten games. The program then shows and ranks the games based on the formula. The page and formula are similar to that of Steam [26], but as said before, for Steam [26], the games shown are only Steam games. As for the program, the game suggestions are based on a wide variety of games including downloadable content (DLC) and modded games. The games suggested are personalized and ranked, but just like any other suggestion system it is subjective to the user, this algorithm narrows it down and has probabilities of whether the user would like it or not. One of the things that could be changed for this algorithm and the others was whether or not to show games that the user can play (games only that the user can play based on the console(s) they have), but after thinking about it the decision was to not do this. This was because it is guaranteed the user has a computer as they need one to access the web application. With this information, users could play most if not all games unless it is a modern-day console exclusive. This is because things such as emulation exist. Users can play almost any game by emulating a console and finding a safe way to find the game they want and if they can't emulate the console that the game is on, it is most likely to be on Steam [26]. Plus a lot of consoles are also backward compatible meaning one console can play older games from

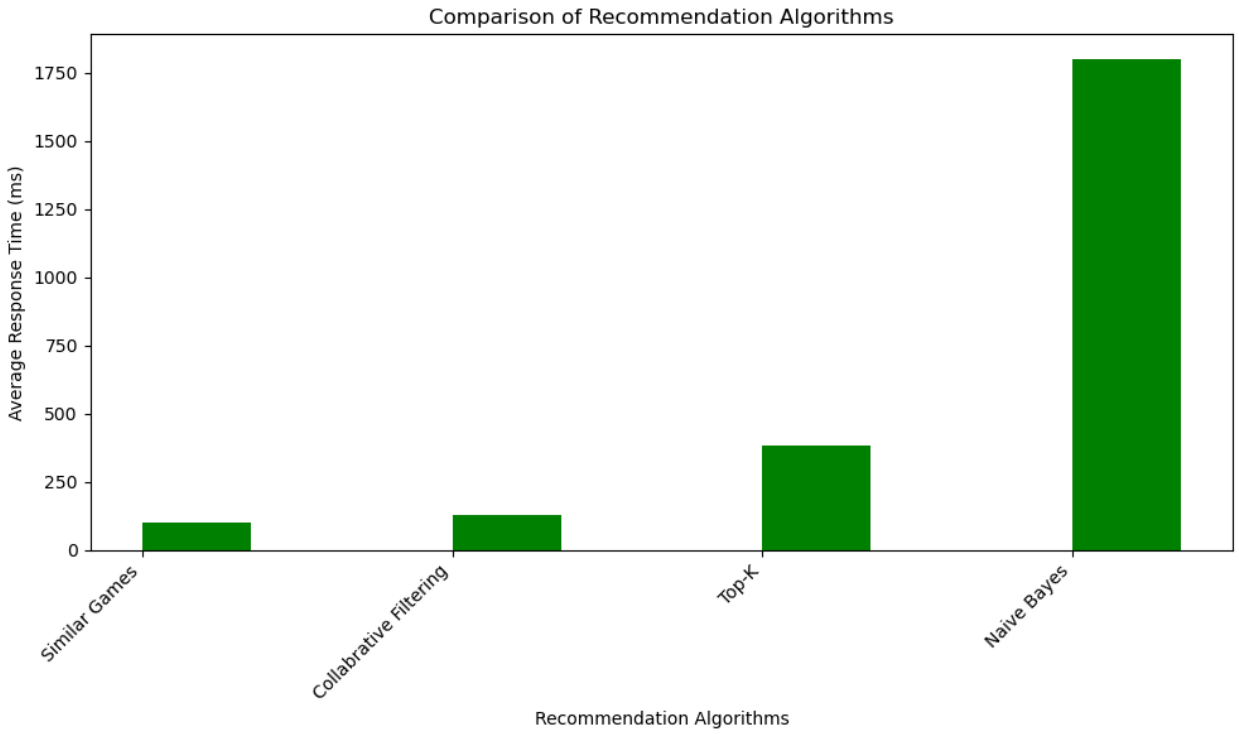
other consoles. So, by keeping games that are on consoles the user doesn't have, there is a highly likely chance they can play it on the computer they have either through emulation or Steam [26] or through backward compatibility from a console they have. Plus the fact that a lot of modern-day consoles and Steam [26] have remakes or ports of games from older consoles. So, with that, it was decided to keep all recommendations and even if they can't play it, they can find out they can't play it and see there are plenty of other suggestions. Also based on the preferences page, this eliminates the problem as it will only show games that the user owns if they only want games they can guarantee to play. As for the preference page, the program would filter and show suggested games based on the user's preference of game mode(s), genre(s), and platform(s) and would also show suggestions that were ranked based on the game database's rating. Unlike the collaborative filtering page, these suggestions aren't as personalized as someone with a preference on game mode(s), genre(s), and console(s) preferences will get the same results as a user with those exact preferences. The only difference is that the program won't show games that the user has already beaten. The only problem with this algorithm which can also show up in other algorithms is that some games that are a collection of other games might show up or some games that have multiple different versions of the same game whether that be a remake or a port can also show up. This is a bit of a problem because if the user has a game in that collection of games they might not be interested in that game as they already have one of those games in the collection or if the game shown is a remake or port of a game that the user has already beaten, the user might not want to buy that game again as they already have it. But, there are a decent amount of users that like those sorts of games. Looking at the real-world data of the users, some of them have the original game and the remake or port, especially if they rated the original pretty highly. This is because some users' favorite games get ported or remade, adding more to their

favorite game almost guaranteeing they will most likely enjoy the port or remake since they liked the original so much. So, with that, the collection of games, remakes, and ports were kept in the data set.

Moving on to the based on user's favorite games page, the algorithm used is also not personalized more so than the based on preferences algorithm since one user who has the same favorite game as another user will get the same result of similar games. However, it was decided to keep it in for more accessibility. This is because if a user is still new to video games and has only beaten one or two games and doesn't know what they like or dislike, they can find games similar to that of the games they played if they enjoyed those games. Especially if that is all they are looking for; a game similar to that they love and have high standards about. Then for the Naive Bayes algorithm [11], the algorithm is like the collaborative filtering algorithm being personalized, but it is an algorithm that isn't used anywhere else like the other suggestion systems like Steam [23]. The algorithm personalizes the user's suggestions by categorizing the user into an average rating category and that shows games that have that average rating. It isn't as personalized as two users can be put in the same average rating category and get the same suggestions, but it is highly unlikely especially if there are a lot of users with different average ratings increasing the number of average rating categories. These suggestions are also ranked on popularity showing that the average is more reliable for the higher-up suggestions as more people rated it giving it a more reliable average.

After looking at the program from a visual point of view seeing that the application was accurate, accessible, clean, and had very well-functioning and personalized algorithms, it was time to look at it from a data standpoint. Using a bit of code to obtain execution time, the data

was used to compare the different algorithms made for the new recommendation system to see which one was the fastest. (Figure 21)




(Figure 21: algorithm comparison table)

Going off the data of the algorithms used in the web application, it was clear to see that if the user wants recommendations fast, they can use the similar games algorithm or collaborative filtering[16] as they have the lowest execution times. Meanwhile Naive Bayes[11] has the highest execution time due to the fact that it is a much more complex algorithm than the rest.

### 5.3 User Feedback

After fixing bugs and errors as well as improving the algorithms to be more accurate, it was time for other users to test it out. Gathering a few people who played video games to test it out and asking what they thought about it. Each started by registering as a user entering games

and consoles and logging in. They then proceed to test out each recommendation method and see their recommendations. Most of them were very happy and satisfied with the recommendations as they were games they never heard of before. All of them added at least one game to their backlog as it caught their eye. As the users were testing out the program, some of them even made comments such as “This is better and more accessible than Steam’s system” or “I would genuinely use this application to log my gaming data and find new games” showing that the application improves on what is already out there and shows that it’s something useful for gamers. It seemed that the collaborative filtering and top-k algorithms were better at creating recommendations as most users added to their backlog games from those recommendation methods while Naive Bayes and games similar to their favorites were not as popular which made some sense as those algorithms didn’t produce as many suggestions as the other two. Also, varying users still had accurate results as users who didn’t have that many games and consoles (users with small data) still showed good and accurate results as users with lots of games and consoles (users with large data). Overall, users were stratified with the recommendations and had a positive outcome as they were glad to say they found a new game they were interested in.

# Discussion

## 6.1 Strengths and Limitations

Overall, the recommendation system proved to be successful. It adeptly provided game recommendations through multiple approaches, ranging from highly personalized to more general methods. The system was effective for both novice and experienced gamers. It includes a comprehensive selection of video games from various consoles, with ratings based on user data to further refine the personalization of recommendations. Additionally, the system offers four distinct recommendation methods, enabling users to choose the approach they believe will yield the most relevant results. The application is designed to be straightforward and user-friendly, with a primary focus on its core function of delivering accurate game recommendations.

## 6.2 Future Research Directions

With all being said, just like anything, the application isn't perfect and can always be improved. As talked about before in the Methodology chapter, the data obtained didn't have everything desired. Even though the data was big and had a decent amount of information, two other things were wanted which would've been how long it takes for the game to be beaten on average and how much the game costs. With this data, it would've changed the based-on preference algorithm to include these as parameters where the user would put a range of how long they want the game to be and what budget the user has on spending a game. Also, it would've added to each recommendation showing the price and how long it takes to beat the game in case users wanted a shorter or longer game or if they wanted a cheap or more expensive game. Finding the price data would be hard, especially for retro games or games that have

paywalls/microtransactions as retro games have no retail price since a lot of them aren't being made anymore and unless the game states it, it's hard to tell if the game has more things you need to pay for once you buy the base game. This could be solved by calculating the average price retro games are going for by sellers and if there is data on games stating whether or not they have paywalls/microtransactions, but from the research, finding a good enough data set for how long to beat that didn't seem possible. As for the average time it takes to beat a game there is a good website called How Long To Beat [12] which does show that information, but couldn't figure out a good way to web scrape and extract that data, and the data found only matched up with a select portion of the games in the data set. Other than that, another thing that could have been done to improve the application was to somehow link their Steam [26], PlayStation, and/or Xbox accounts to the user's profile to add the games from those accounts to the user's profile. The only problem would be that those games wouldn't be rated and they couldn't be determined if they beat those games or not, but it would be a useful way to add games instead of the user having to manually enter each game and rating. However, with the ability of the user being able to enter the games they want to enter, the user can manipulate the data by only adding games they liked to increase their chances of getting a suggestion they like instead of having the list of all the games they beat which include games they may not enjoy as much. The only downside is that those games they didn't like as much that the user didn't enter could pop up as a suggestion instead. Lastly, the other thing to change/add is to have the ability to update the game data set as new games and updates are always being added and thus they should also be added and updated to the data. Doing this automatically seemed impossible, unless the data was loaded from the API every time, but that would take a long time. So, instead, the other option was to manually update the data by receiving it from the API every once in a while.

With all the errors and bugs fixed and testing all the algorithms, the end product was a pretty reliable suggestion system. One that has some personalized algorithms such as the user similarly and Naive Bayes algorithms [11] as well as some algorithms that are just based on the data which were based on the user's preferences or the user's favorite games. The algorithms can be compared to other suggestion systems such as Steam, but with the data obtained, the recommendations have more variety and eliminate all games that the user has beaten as well as show if the user has the game in their backlog or not.

# Conclusion

After identifying and resolving all bugs and errors, a functional program was achieved with minimal issues. The data was meticulously cleaned and organized, the algorithms were tested and optimized for accuracy and efficiency, and the web application was designed to allow users to register, log in, edit their profiles, and, most importantly, receive personalized recommendations while having easy access to all functionalities. This project was both challenging and educational, representing the first experience in developing such a large-scale web application and the second time undertaking a significant project. Nonetheless, there remains potential for further improvement, as additional features could be implemented given more time and resources.

## 7.1 Learning Process

This project was a big struggle. Only doing one other big project like this with the senior project with the undergraduate program. But unlike the senior project which was a simple video game that was very long and tedious with the help of a video game engine, this project was completely different being less tedious and more research testing, and tuning as it has a lot more moving parts to it. It was the first time making such a big personalized web application with a little bit of experience prior in the undergraduate program, and it was the first time using the backend FastAPI [7]. With this project, a lot more about web applications was learned and a lot more experience with web applications was obtained as well as FastAPI [7] and React [24]. More knowledge and experience with algorithms and being able to convert algorithms into Python code and implement them for the data sets were learned. With the data sets, more experience in scraping the web for data was obtained as well as working with APIs and cleaning

and preparing data to have the ability to work well with the algorithms. The project took a bit longer than expected because of the lack of experience with web applications and big projects like this, but now with this project, more comfortability with big projects and with web applications was obtained.

### 7.2 Summary of Findings

Personalizing and creating an accurate recommendation system proved challenging, as many recommendations are inherently subjective and there is no guarantee that any single game will appeal to everyone. Just as with any form of media, there will always be individuals who do not enjoy certain games. However, the developed algorithms and web application effectively create a robust suggestion system. Given the four different algorithms integrated into the application, it is highly likely that at least one will provide recommendations that users will find appealing, particularly those based on top results, which are more likely to align with user preferences.

Additionally, identifying and resolving bugs and errors presented a significant challenge due to the extensive data and numerous user inputs that could potentially disrupt the code. Most issues were addressed, and the web application was refined to be simpler, more accessible, and user-friendly, ensuring a positive user experience.

### 7.3 Contributions to the Field

The application as a whole can be considered better than the other video game suggestion systems such as Steam [26], IGDB [14], and RAWG [23] as it has more variety of both games and algorithms as well as more personalized suggestions and more customization for the user's

profile. As the web application uses four different algorithms all using different forms of suggesting and providing different results with low mean squared error percentages and ranked by probabilities. Also with a lot of different options for the algorithms that aren't as personalized that find games based on preferences or by user's favorite games. The other suggestion systems have some of these algorithms, but they don't have the variety of games or they aren't as personalized, or don't have all the algorithms that the application has. The web application only is good for suggesting games and personalizing users as the other suggestion systems do other things such as be the database for games, however, even though the application does one major thing, and that one thing alone is mostly to suggest games, it exceeds at that and does a better and more efficient job of that than the others. With that, it also keeps it simple as it only has one job and does that job well by also not having anything else to overwhelm or confuse the user.

### 7.4 Final Thoughts

Overall, this project involved substantial effort, including the acquisition and application of numerous new techniques on a large scale. Although the process was lengthy, the outcome is commendable. While the system is not perfect and has room for further enhancement, it effectively surpasses existing recommendation systems by providing personalized suggestions and diverse methods for recommending games to users. The application is designed with simplicity in mind, offering an intuitive interface for users to navigate and access its features. With the algorithms integrated into this application, it is anticipated that users will find it easier to discover games they enjoy, resulting in a more satisfying and worthwhile investment of their time and money.

# List of References


[1] Activision.com. Accessed: February 20, 2024. [Online.] Available: https://www.activision.com/?utm_source=404&utm_medium=redirect&utm_campaign=122222

[2] AppleMusic.com. Accessed: February 20, 2024. [Online.] Available: https://music.apple.com/us/browse

[3] Bethesda.net. Accessed: February 20, 2024. [Online.] Available: https://bethesda.net/en/dashboard

[4] Blizzard.com. Accessed: February 20, 2024. [Online.] Available: https://www.blizzard.com/en-us/

[5] ChromeDriver. Accessed: June 20, 2024. [Online.] Available: https://developer.chrome.com/docs/chromedriver

[6] EA.com. Accessed: February 20, 2024. [Online.] Available: https://www.ea.com/

[7] FastAPI.com. Accessed: April 25, 2024. [Online.] Available: https://fastapi.tiangolo.com/

[8] Flask.com. Accessed: April 5, 2024. [Online.] Available: https://flask.palletsprojects.com/en/3.0.x/

[9] GameFreak.co.jp. Accessed: February 20, 2024. [Online.] Available: https://www.gamefreak.co.jp/

[10] *Hacking the Thesis*. (n.d.). The Ohio State University. Retrieved February 16, 2022, from https://u.osu.edu/hackingthethesis/managing-stuff/your-content/outline/

[11] Hand, David & Yu, Keming. (2007). Idiot's Bayes: Not So Stupid after All?. International Statistical Review. 69. 385 - 398. 10.1111/j.1751-5823.2001.tb00465.x.

[12] HowLongToBeat.com. Accessed: March 3, 2024. [Online.] Available: www.howlongtobeat.com

[13] Hulu.com. Accessed: February 20, 2024. [Online.] Available: https://www.hulu.com/welcome?orig_referrer=https%3A%2F%2Fwww.google.com%2F

[14] “IGDB: Video Game Database API.” IGDB.com. Accessed: February 20, 2024. [Online.] Available: www.igdb.com/api.

[15] Instagram.com. Accessed: February 20, 2024. [Online.] Available: https://www.instagram.com/

[16] S. Manakkadu, S. P. Joshi, T. Halverson and S. Dutta, "Top-k User-Based Collaborative Recommendation System Using MapReduce," 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, 2021, pp. 4021-4025, doi: 10.1109/BigData52589.2021.9671395. keywords: {Web services;Collaborative filtering;Scalability;Clustering algorithms;Cluster computing;Big Data;Filtering algorithms;Recommender systems;Collaborative filtering;MapReduce;Social networks;MovieLens}

[17] Metacritic.com. Accessed: June 20, 2024. [Online.] Available: www.metacritic.com/.

[18] MicrosoftStore.com. Accessed: February 20, 2024. [Online.] Available: https://www.xbox.com/en-US/microsoft-store

[19] Netflix.com. Accessed: February 20, 2024. [Online.] Available: https://www.netflix.com/

[20] Nintendo.com. Accessed: February 20, 2024. [Online.] Available: https://www.nintendo.com/us/

[21] PlayStationStore.com. Accessed: February 20, 2024. [Online.] Available: https://store.playstation.com/en-us/pages/latest

[22] PostgresSQL.org. Accessed: April 2, 2024. [Online.] Available: https://www.postgresql.org/

[23] RAWG.com. Accessed: February 20, 2024. [Online.] Available: www.rawg.io

[24] React.dev. Accessed: April 28, 2024. [Online.] Available: https://react.dev/

[25] Spotify.com. Accessed: February 20, 2024. [Online.] Available: https://open.spotify.com/

[26] Steam.com. Accessed: February 20, 2024. [Online.] Available: www.store.steampowered.com/

[27] Tailwind.com. Accessed: June 8, 2024. [Online.] Available: https://tailwindcss.com/

[28] TikTok.com. Accessed: February 20, 2024. [Online.] Available: https://www.tiktok.com/en/

[29] *What Is a Recommendation System?* NVIDIA Data Science Glossary. Accessed: February 16, 2024, from www.nvidia.com/en-us/glossary/recommendation-system/..

[30] YouTube.com. Accessed: February 20, 2024. [Online.] Available: https://www.youtube.com/

# Appendices

- Code used to obtain game data set from IGDB - Thesis Start.ipynb
- Code used to clean up the game data set - video_game_data_clean_up.py
- Code used to make example user data set - Thesis Start.ipynb
- Code used to test collaborative filtering algorithm - Thesis Start.ipynb
- Code used to test based on preferences algorithm - ML model.ipynb
- Code used to scrap real-world user data - Real User Data and Naive Bayes.ipynb
- Code used to test Naive Bayes algorithm - Real User Data and Naive Bayes.ipynb
- Code used to find execution time and accuracy of the algorithms - Execution time.ipynb
- Code used to transfer the data sets into SQL Postgres - PostgresSQL.py
- Code used for the backend of the web application - main.py
- Code used for routing - App.js
- Code used for authority/user security - AuthContext.js
- Code used for the frontend of the edit profile page - EditProfilePage.js
- Code used for the frontend of the preference page - GameModePage.js
- Code used for the frontend of the home page - HomePage.js
- Code used for the frontend of the login page - LoginPage.js
- Code used for logout button - LogoutButton.js
- Code used for the frontend of Naive Bayes page - NaiveBayesPage.js
- Code used for the frontend of the profile page - ProfilePage.js
- Code used for the frontend of the register page - RegisterPage.js
- Code used for the frontend of similar games page - SimilarGamesPage.js

- Code used for styling - styles.css
- Code used for tailwind - tailwind.css
- Code used for the frontend of the collaborative filtering page - UserSimilarityPage.js