CSC111 Project Report: Steam Game Recommender

Cheng Peng, Mikhael Orteza, Ari Casas Nassar, Muaj Ahmed

Monday, April 3, 2023

Problem Description and Research Question

- Overview: Steam is a PC gaming platform used by millions across the world. It was launched in the early 2000s by Valve, which has made famous games such as Counter-Strike: Global Offensive and Dota 2. Players can download games from the 30,000+ options on Steam (Dimuro, 2021). Each game on the Steam store can be classified using its tags, such as genre, user reviews, and price.
- Gaming is a hobby many people have for different reasons, such as alleviating boredom, getting rid of stress, or simply because of how fun it is. Whether the gamer is new or experienced, the hunt for new games is a task all gamers have. Experienced gamers will have trouble finding games they like that they haven't played, whereas new gamers will have trouble finding the "perfect" game among the many Steam games available. Steam has an interactive recommendation system to help gamers find new games they'd like. Steam recommends new games for a player based on what the player has played and compares it with other players who have similar playing habits. The basic idea is that if the other players have a game that the player using the Steam recommender doesn't have, Steam will recommend that game to them ("Steam News," 2020). Though this system is sound, one downside is that for a player who hasn't played many games, Steam would consider them to enjoy the genres of those few games mainly. In contrast, it is possible that they've just begun gaming and haven't fully explored all the possible genres.
- Project Goal: **To create an interactive program that recommends users games based on their preference of genre, games they've already played, and the amount of money they're willing to spend.**
- Side note: The TA who graded and read our proposal suggested that we needed to be clearer about how we were taking advantage of the graph data structure to make recommendations. In our proposal, we suggested creating a graph of nodes for every steam game in a given data set. However, we suggested looking at every node in the graph and finding the nodes with the highest meta scores based on their various characteristics. After receiving the TA's feedback, we realized the flaw in our proposal and made appropriate changes to how our program would be made.

Datasets

The dataset is called 'Game Recommendations on Steam' by Anton Kozyriev. The dataset consists of multiple CSV files. Out of all these files, the CSV named games.csv will be used. In this CSV file, each row contains thirteen columns. However, our program will use the title, price_final, app_id, and positive_ratio columns. However, games.csv does not have a column that does not contain the genres of each game, which is crucial for providing recommendations in our program based on the user's preferences. Thus, we also decided to use a JSON file in the data set called "games_metadata.json," which contains the genres of each game in games.csv along with their associated app_id. To use the file named games_metadata.json, we imported the JSON library to easily get a dictionary form of data for each line in the JSON file. As a result, the dictionary will have three keys: the game's app_id, description, and tags. Out of these keys, only app_id and tags will be used. The app_id will allow us to combine the data from the CSV file with the JSON file. Furthermore, the value associated with the tags key in the dictionary will provide us with every game's genre.

Attached below is some sample data with a few columns removed.

title	*positive_ratio*	*price_final*
Call of Duty: World at War	92	19.99

Link to dataset: https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam

Computational Overview

We used a dataset of various steam games and their characteristics to compute the game's recommendation score. To provide recommendations to the user, every steam game was scored based on the user's inputs into the user interface. The inputs include all the steam games the user has played or wants the recommendations to be based on, the maximum price the user is willing to pay for a game, and all the genres the user wants the recommended games to have. After receiving the user's preference inputs, a node graph that refers to a specific steam game from the data set is created. Edges are formed between two nodes if one of the nodes is a user-inputted game and both nodes share at least one genre. The game_graph.py file is where we created our graph and performed computations. When the generate_game_graph function is called in the runner of game_graph.py, a graph is formed with a certain number of nodes that the user can change. The user can also change the number of games they want to recommend to them by changing the num_games_recommended variable in the runner function. The reason we gave the user to choose the number of nodes is because as this number increases, the run time of the code does too so this feature makes it better for the user.

After the nodes and edges are formed, the program computes each game's score. How a game's score is computed depends on whether the user has inputted any games. If the user has inputted any games, a game's recommendation score will be based on its number of neighbours, price, genres, and positive ratio rating. First, 50% of the total score will be based on the two ratios multiplied together: the game's price compared to the highest-priced game and the game's positive ratio compared to the highest-positive ratio. Next, 30% of the score is based on the number of neighbours a node has compared to the total number of possible neighbours, which is the number of games the user has inputted. The neighbours are checked because games that have more neighbours will be more similar to the games that the user has recommended. Regarding the user node's neighbours, another 10% of the score is computed by the average positive ratio of all the games in each neighbour node. Thus, if the game is connected to higher positively ratioed user inputted games, they will also be more likely to be a better recommendation. Finally, 10% of the score will be based on how many genres of each game are similar to the user's all the genres of the user's inputted games. With all these parts of the score computed, they are added together and are assigned as the game's score only if the game's price does not exceed the user's budget, and has all the user's inputted genres. If any of these conditions are met, the game score will be set to 0.0. On the other hand, if the user did not input any games, then the user must have only inputted various genres that a recommended game must have. In this scenario, 60% of the total game score will be based on the game's positive ratio and the price, using the same computations mentioned above. However, 40% of the score will be based on how many genres of the game coincide with all the genres the user has inputted. These two parts of the score are added together, and the sum is assigned as the game's score if the price doesn't exceed the user's maximum budget. If the price does exceed the budget, the score is zero.

Now that each game has its respective score, the top x scored games (default is 5) must be found. This is done in two ways, depending on if the user inputted any games. If the user did input games to base recommendations on, a collection of games is created by adding all the neighbours of the game nodes in the graph whose game is user-inputted. All the neighbours are possible recommendations for the user since they share at least one genre with the user-played game, so they have some similarities. Nonetheless, with the collection of possible recommended games, the program creates an accumulator that stores the games that will be recommended and added onto the accumulator by iterating through the collection of possible recommended games and adding the highest-scored game into the accumulator. This process is done until the accumulator has reached the number of top games that need to be found or the collection of possible recommended games is empty. Note if a possible recommended game is a game that the user has inputted, it will not be accounted for. Nonetheless, now that the top games are found, the list is sorted in descending order using an iterative insertion approach. The list is returned, and the games are displayed in the user interface.

For the visualization of our program, we used tkinter. All the interactive elements of our program were written using

tkinter and are defined under the user_interface.py file. The tkinter library was the best library for us to use for visualization because it allowed us to make a minimalistic application that is easy to use. The interface of Tkinter comprises individual widgets, which are Python objects instantiated from other classes like ttk.Frame, ttk.Label, and ttk.Button ("tkinter," n.d.). With these widgets along with others like Listbox, we were able to display the results of our program to the user.

Instructions for Users

- **Python libraries to install:** Please see requirements.txt for this information.
- **Downloading Datasets:**

  Download the zip folder named 'datasets' which has been uploaded on MarkUs. This file includes the csv file and the json file. Then unzip the file and extract its contents to pycharm. The image below will show how the screen should look. Note that there should be a folder called dataset that has the csv file and json file, and there should be 4 files that are on the same level as the dataset folder.

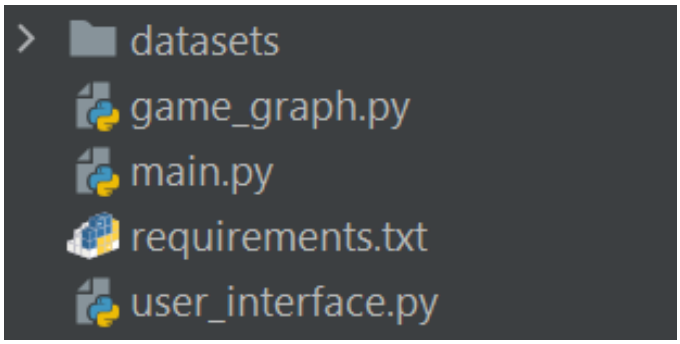

Figure 1: This is what the files should look like in Pycharm.

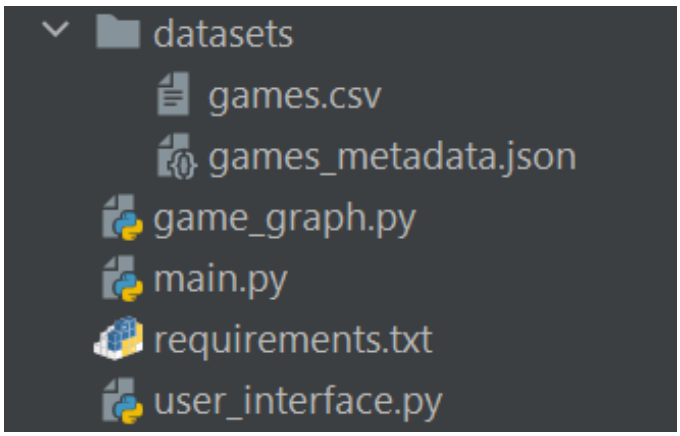

Figure 2: This is what the it should look like with the dataset folder expanded.

- **Running the Program:**

1. Open main.py and run the file in console. A window will appear which introduces the user to the steam game recommender. The user will be asked to input game ids of steam games they've already played. Adding a game is optional and if the user wants, they can click done to move on to the next step. On the bottom, there is a list of valid steam game ids that the user can input. The list is sorted by the game name in alphabetical order. Open main.py and run the file in console. A window will appear which introduces the user to the steam game recommender. The user will be asked to input game ids of steam games they've already played. Adding a game

is optional and if the user wants, they can click done to move on to the next step. On the bottom, there is a list of valid steam game ids that the user can input. The list is sorted by the game name in alphabetical order.

2. Next, the user can select up to six genres they're interested in. The user must select at least on genre.

3. Then, the user needs to input the maximum amount of money (in USD) that they are willing to spend. This helps filter out games that are not in the user's budget.

4. Finally, the program will take a few seconds, if not a few minutes (this depends on the number of nodes the user decided to make in the runner function of game_graph.py), and then display the top x number of recommended games where x is the number of games the user wanted to be recommended to play. The steam link is also provided so if the user clicks on the link, they will be redirected to the steam game page to find out more information about the game.

Attached below are what to expect at each step:

Figure 3: Step 1

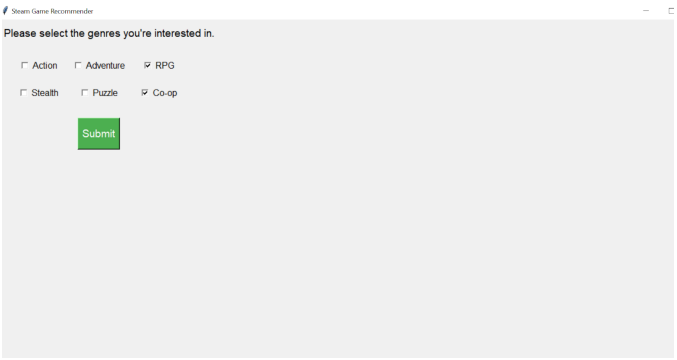

Figure 4: Step 2

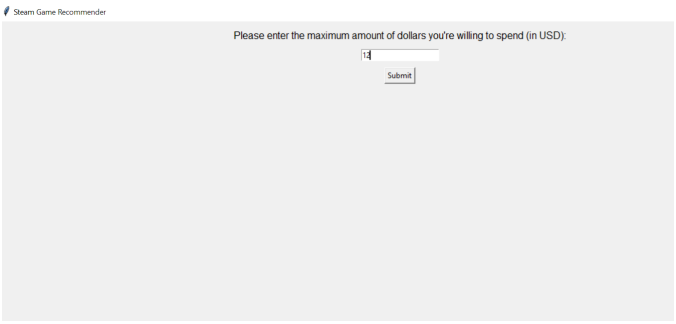

Figure 5: Step 3

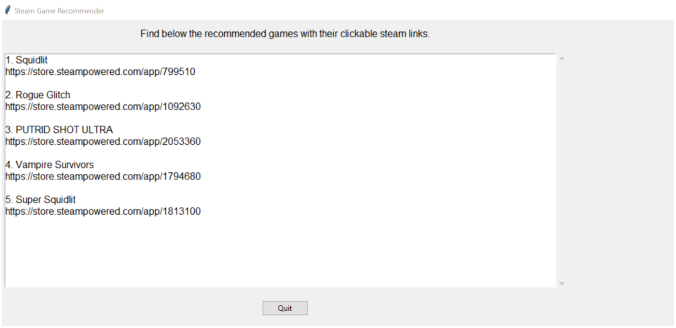

Figure 6: Step 4

Changes to the Project Plan

As mentioned in the introduction, we had to adjust how our program would be made after receiving the TA feedback. In our original approach, we planned to create a graph consisting of game nodes and basing recommendations solely on genre inputs. Edges would have been formed between two nodes if they shared several genres. With this graph idea, we planned on creating various graphs where the genre edge requirement changed and computing the highest scores from all the nodes in the graph based on the number of genres that coincided with the user-inputted genres, the price, and the positive ratio. However, the program did not take advantage of the graph data structure with this approach. It was pointless to create edges between nodes depending on the edge genre requirement if the edges were not taken advantage of. Thus, we changed the project by allowing users to input games they want recommendations to be based on. With these games in the graph, we can take advantage of the fact that edges will be formed between a user-played game node and another node if they share any genres. Furthermore, if a game has more neighbours, meaning that it is similar to more games the user has played, it also helps to find similar games that the user likes. Thus, this idea has been implemented into our program to take advantage of the graph structure. Changes to the graph structure and score computation also influenced changes to the user interface since we needed to account for user inputs of genres, games, and budget prices. Furthermore, since we made it so that the user will need to input the game ids for each inputted game, we decided to show all the valid game ids and their game title in the user interface. Thus, the user can find valid games to base recommendations on easier.

Program Results Analysis and Interpretation

- Conclusion: Our goal in this project was to recommend new games to the user based on their preference of genre, budget, and past games they've played. Our program successfully does recommend games as described in our project goal. One aspect of our program we are proud of is how easy it is to use and understand. There are many interactive components of our program, and we feel that this made the user experience better. In summary, our program does what we wanted it to do, but there is room for improvement in our algorithms and limitations, which we will discuss below.

- Limitations:

  - Our dataset isn't completely up to date like Steam's library is, so it's possible that a game that came out isn't in our csv file.
  - The csv file was too large (about 46 thousand rows), causing the graph creation to take too long. Thus, we had to use a certain portion of the csv file to prevent long running times. However, using a certain portion also led to another problem in the program. When an invalid game was inputted (a game that isn't in the portioned csv file), it was initially recognized as valid since the user_interface module saw every single game id in the unedited csv file as valid. However, since only a portion of a file was used in the graph creation, the graph had no user_nodes (a dict for storing all the user nodes), which affected the score computation since the program used the user-inputted game computation approach where user_nodes needed to be filled. Thus, we had to adjust the csv runner function only to return a list of a certain length. This certain length is what the user inputs in the runner function that is in game_graph.csv.

- Next Steps:
  - Implementing a more personalized recommender: We only used three attributes to determine what games to recommend to the user. By using other attributes such as popularity and operating system in addition to the ones we've used, the game recommendations to the user will be better. Furthermore, if the user is more lenient about spending more money and does not solely want cheap games, we could have made it so that the program does not account for prices. While testing the code, some of the recommended games tend to be cheaper since lower prices influence higher game scores in our program, which might not be the preference for every user. Also, we could have used the recommendations.csv to make a more in-depth recommendation by looking at all the user reviews for the games and considering how good the review is and the number of hours the user spent on the game. This approach would have also helped when using the positive ratio attribute for each game. This is because games with a low number of reviews but an extremely high rating might not always be necessarily good.
  - Algorithm efficiency: It's evident that there needs to be work done to improve the run time of our functions, specifically those used in generate_graph, which is a function defined in the game_graph.py file.
  - Improving user interface: Though our program is easy to use, it does lack visual appeal. Using a library like pygame, we could make our program visually appealing. Also, instead of having the user to scroll down the list of possible game ids they can input, we can work on adding a feature to filter out the list of possible games by adding a search bar the user can type into.

References

Claudia D. (2021, July 26). *What is Steam? What you need to know about the online video game platform.* Pennlive. https://www.pennlive.com/life/2021/07/what-is-steam-what-you-need-to-know-about-the-online-video-game-platform.html

"Json — JSON Encoder and Decoder." *Python Documentation*, docs.python.org/3/library/json.html.

Kozyriev, A. (2023, February 28). *Game Recommendations on Steam.* Kaggle. https://www.kaggle.com/datasets/antonkozyriev/game-recommendations-on-steam

Steam News - Introducing The Steam Interactive Recommender - Steam News. (2020, March 18). https://store.steampowered.com/news/app/593110/view/1716373422378712840

tkinter — Python interface to Tcl/Tk. (n.d.). Python Documentation. https://docs.python.org/3/library/tkinter.html