Project Group 5
Johan Söderström
Axel Nilsson
Pavlos Papadopoulos

johan.soderstrom.9461@student.uu.se
axel.nilsson.3477@student.uu.se
pavlos.papadopoulos.0827@student.uu.se

Common Steam Games

February 2021

1DL201

Syllabus for Program Design and Data Structures

Introduction

Steam is a gaming platform where people from all over the world come together to play games. The platform hosts roughly thirty thousand games and over a hundred million users[1], which has made online gaming rather convenient due to the vast amount of games and the large user base. But despite this larger amount of games on Steam, finding a game to play together with your friends has always proved to be quite difficult. Therefore, we decided to create a program that, although would not eliminate the problem, would make it easier to find games to play with your friends. We figured that one great way to help people, especially those with large libraries of games, would be to actually present all the games owned by everyone in this party of people that would want to play together. As previously mentioned, this would not entirely remove the hard decision of finding games to play together with your friends, but it would make it easier, as it gives you a list of games available for you to choose from without having to spend money buying the game you decide upon. Additionally, this list would not necessarily show you a list of games that everyone would want to play but rather a list of available games that everyone can play, without having to spend money. Knowing this, we were still interested in creating a program that would search for common games in our Steam game libraries, as there is currently no official way of doing this within Steam. We thought this program would be fitting, as we initially wanted to create a program that would make use of a public API (Application Programming Interface) and gather information from the internet.

Objective

Our objective is to create a program that takes the steam64ID of two or more people and then returns a list with all the games those people have in common in a comprehensible manner.

[1] https://store.steampowered.com/about/

Background

We decided to create this program because we were interested in becoming more familiar with creating programs that retrieve data from the internet. We also wished to become more knowledgeable in parsing JSON-files and presenting data gathered from JSON-files in a readable and well-kept manner. Additionally, we hoped to create an efficient workflow and become accustomed to working together with a Github repository, as we figured that it would prove useful in our future group projects.

Steam

Steam is a digital gaming and online distribution platform launched by Valve in 2003.[2] In Steam's *Store Page*, users can purchase digital copies of various video games which will then be linked to their particular account, rather than the computer itself. Additionally, Steam has a *Community Page*, which acts as the social section of the platform where users can add each other as friends, send messages to each other and write reviews for the games they have played.[3][4] Users can also create modifications and skins for various games and then release them in their official workshop, for other users to download and enjoy.

Implementation

Steam has multiple public APIs that give out different kinds of information. For example, Steam has an API that returns the latest news for a game if you provide it with the app ID of the game you want news from (GetNewsForApp). Steam also has an API that returns the global achievements overview of a specific game in percentages if you provide it with the ID of the game (GetGlobalAchievementPercentagesForApp). In order to specify arguments to an API all you have to do is to add to its URL, which is fairly simple. The program that we created utilizes for the most part the public web API that allows us to gather information on the games owned by the users we provide the steam64ID of (GetOwnedGames). We also make use of another Steam API to gather the usernames of the people we are comparing games for, in order to make it somewhat more comprehensible (GetPlayerSummaries).

---

[2] https://www.gamespot.com/articles/gdc-2002-valve-unveils-steam/1100-2857298/
[3] https://store.steampowered.com/about/
[4] https://en.wikipedia.org/wiki/Steam_(service)#User_interface

Using the program

User requirements

Public Steam settings

In order for the Steam API to be able to extract data from a Steam profile, that profile is required to be public. If your profile is private, you can make it public by going to your profile and pressing “Edit Profile” on the right. Once you press it, look for the privacy settings on the left side of the screen. It should be located somewhere on the bottom. Once in the privacy settings, you can turn your profile public by clicking the highlighted and underlined text followed by “My Profile: ”.

Steam64ID

In order to garner the relevant information on each Steam user, the program requires its user to enter their Steam64IDs. There are multiple ways of finding your steam64ID, however the easiest way would be to use an external Steam ID converter, such as steamid.io, that takes a link to your Steam profile and returns a list of all your Steam IDs. It is also possible to get a hold of your steam64ID through Steam by going to your profile, right clicking and opening the source code for your profile page. A notepad should appear where you can search for “g_steamID” which should be followed by your ID.

GHCi requirements

Since we do not have an executable that one can click and simply make the program work, the user has to install some packages to make it work. Firstly the user is required to have GHCi and the Stack tool installed on their computer. Secondly the user needs to have the required packages installed on their computer, however simply downloading the project through GitHub should also download the packages. This means that you only have to download the packages if you are rewriting the program from scratch. Lastly, an internet connection is required in order for Steam’s web API to be able to gather data.

Running the program

In order to run the program, all the user is required to do is to compile the ‘Main.hs’ file in Stack GHCi by simply typing ‘Stack ghci’.Do recognize that issues may occur for various reasons if you type ‘Stack ghci’ which will in some cases resolve themselves if you instead only compile the main function through stack by typing ‘Stack ghci ./app/Main.hs’. Do also make sure that you are in the right place when you are running ‘Stack ghci’. Once all the files are compiled correctly, the user only needs to run the function ‘main’ by simply typing ‘main’ in the interactive environment. The IO action will then prompt the user to “enter a valid Steam64 ID to a PUBLIC Steam profile”, once a valid ID has been entered, the program will print a list of the Steam user’s owned games and ask you if want to compare the following lists or if you want to continue adding users to the comparison list. If the user wants to add more Steam users to the comparison list, they are prompted to enter anything other than ‘True’ in the terminal. The program will then ask for a valid Steam64 ID once again and repeat this process until the user enters ‘True’. When ‘True’ has been entered, the program prints a list of games that all Steam users in the comparison list have in common.

For example, entering the following IDs:
76561198046588035
76561198068497293
76561198072508175
Returns this list of games:
["Team Fortress Classic","Garry's Mod","Portal","Left 4 Dead 2","Portal 2","Counter-Strike: Global Offensive","PAYDAY 2","Risk of Rain 2","Among Us","Total War: SHOGUN 2"]

Creating the program

Tools

The tools we used for this project were few, not including all the essential ones like a computer or internet. We used a text editor with a terminal, for example Visual Studio Code or Emacs with Haskell mode, to write and run code.

Additionally, the Haskell tool Stack was used for easily installing and running packages in Haskell. It was more convenient than Cabal, since Cabal projects are not as easy to work with on different computers, which was one major requirement for this project. Additionally, there were several useful libraries and packages that were included in the Stack repository, which facilitated the package and library management.

Packages and libraries

Aeson - The program utilizes the Aeson package in order to gather the data from the web API and to then parse said data. The parsing ensures that merely the game titles of the user's owned games are extracted, since that is relevant data.
Data.List - Since the owned games would be in a list, the list library was imported in order to make use of its various features, such as intersect, which takes a list of lists and returns the elements that all lists shared.

Structure

Steam's Web API and Security

Steam's Web API can be used to retrieve information on Steam users via an HTTP request,[5] such as a list of their owned games and for how long they have played them. It can also retrieve data like the user's account name, given location and their friends on the platform.

[5] https://partner.steamgames.com/doc/webapi_overview

The API has several features which are divided up into various interfaces. They each contain a certain category of information, which is convenient later when parsing. For instance, the GetOwnedGames interface retrieves merely a list of owned games of a user. An API key is obtained via Steam account, which means that it will remain linked to said account. The key should not be shared with others since scammers have been known to use these keys in order to steal items from people's accounts.[6]

Parsing

To make use of the API's data, the JSON-file had to be parsed. Parsing means interpreting the JSON-files information in a way that it can later be used in a specific programming language, which in this case is Haskell. The JSON-file may contain an unfamiliar data type, which means that in order to obtain usable information, the program needs to know the structure of the JSON-file in order to pick out, for instance, a specific value such as the title of a game. In order to send requests for retrieving data to the Steam API, the Network.HTTP.Conduit library was used, specifically the simpleHttp function.[7] As for parsing this obtained data, the Aeson package was used for writing the custom data types and to declare the To- and FromJSON instances.[8]

Data Types

In order to properly parse the JSON-file from the API, we had to write our own data types that resemble the structure of the file. For the list of games, we had to parse the GetOwnedGames interface of the API.[9] Since the file contained a list of games, we began with declaring a data type called "Game". It contains information such as the game's name, for how long the user has played it, etcetera.

```
data Game = Game
        { appidGame :: Int
        , nameGame :: Text
        , playtimeForeverGame :: Int
        , imgIconURLGame :: Text
```

[6] https://steamcommunity.com/groups/lootfarm/discussions/0/1608274347722600472/

[7] https://hackage.haskell.org/package/http-conduit-2.3.8/docs/Network-HTTP-Conduit.html

[8] https://hackage.haskell.org/package/aeson-1.5.6.0/docs/Data-Aeson.html

[9] https://developer.valvesoftware.com/wiki/Steam_Web_API#GetOwnedGames_.28v0001.29

```
, imgLogoURLGame :: Text
, playtimeWindowsForeverGame :: Maybe Int
, playtimeMACForeverGame :: Maybe Int
, playtimeLinuxForeverGame :: Maybe Int
, hasCommunityVisibleStatsGame :: Maybe Bool
, playtimeMACForeveRGame :: Maybe Int
, playtime2WeeksGame :: Maybe Int
} deriving (Show)
```

The JSON-file stores each game inside of another data type called “games”, which also contains the amount of games inside a variable called game_count, we named this type “UserGamesResponse”. Finally, this data type is used in a third one, GamesResponse, that contains one UserGamesResponse.

```
data UserGamesResponse = UserGamesResponse
    { gameCount :: Int
    , listOfGames :: [Game]
    } deriving (Show)
```

In order to parse information about the user’s profile, we had to write separate data types for the GetPlayerSummaries interface.[10] Although the method for obtaining the relevant information is the same as with the GetOwnedGames.

Instances

Another requirement for properly parsing a JSON-file are the FromJSON instance declarations. These instances are necessary for the Aeson function eitherDecode to operate properly. The instances tell the decode function which value in the JSON-file is supposed to be associated with which value in its corresponding data type. The instance declaration for the Game data type looked like this.

```
instance FromJSON Game where
    parseJSON (Object v) = Game
    <$> v .: "appid"
    <*> v .: "name"
```

[10] https://developer.valvesoftware.com/wiki/Steam_Web_API#GetPlayerSummaries_.28v0002.29

```
<*> v .: "playtime_forever"
<*> v .: "img_icon_url"
<*> v .: "img_logo_url"
<*> v .:? "playtime_windows_forever"
<*> v .:? "playtime_mac_forever"
<*> v .:? "playtime_linux_forever"
<*> v .:? "has_community_visible_stats"
<*> v .:? "playtime_mac_forever"
<*> v .:? "playtime_2weeks"
```

Like the data types, these declarations must have the same structure as the JSON-file and also the same data type as in its corresponding data type. For example, '<*> v .: "name"' declares that the name of a game is of a *value*. Whilst, '<*> v .:? "has_community_visible_stats"' declares that has_community_visible_stats is of a *Maybe value*. This is due to the fact that if the user has set, for example their community stats to private, then the API is unable to retrieve that data. If that is the case, then its value becomes *Nothing* and the parsing can continue. Otherwise, the parsing would fail and an error would be returned instead, regardless of which value we are looking to extract. With a correctly written data type and FromJSON instance, the decode function can easily convert the value in the JSON-file to a Haskell comprehensible value, such as the name of a game, which will be converted into the Text data type.

Functions

fromJSON

There is one function for each piece of information that is retrieved from the API, for instance the

Intersect

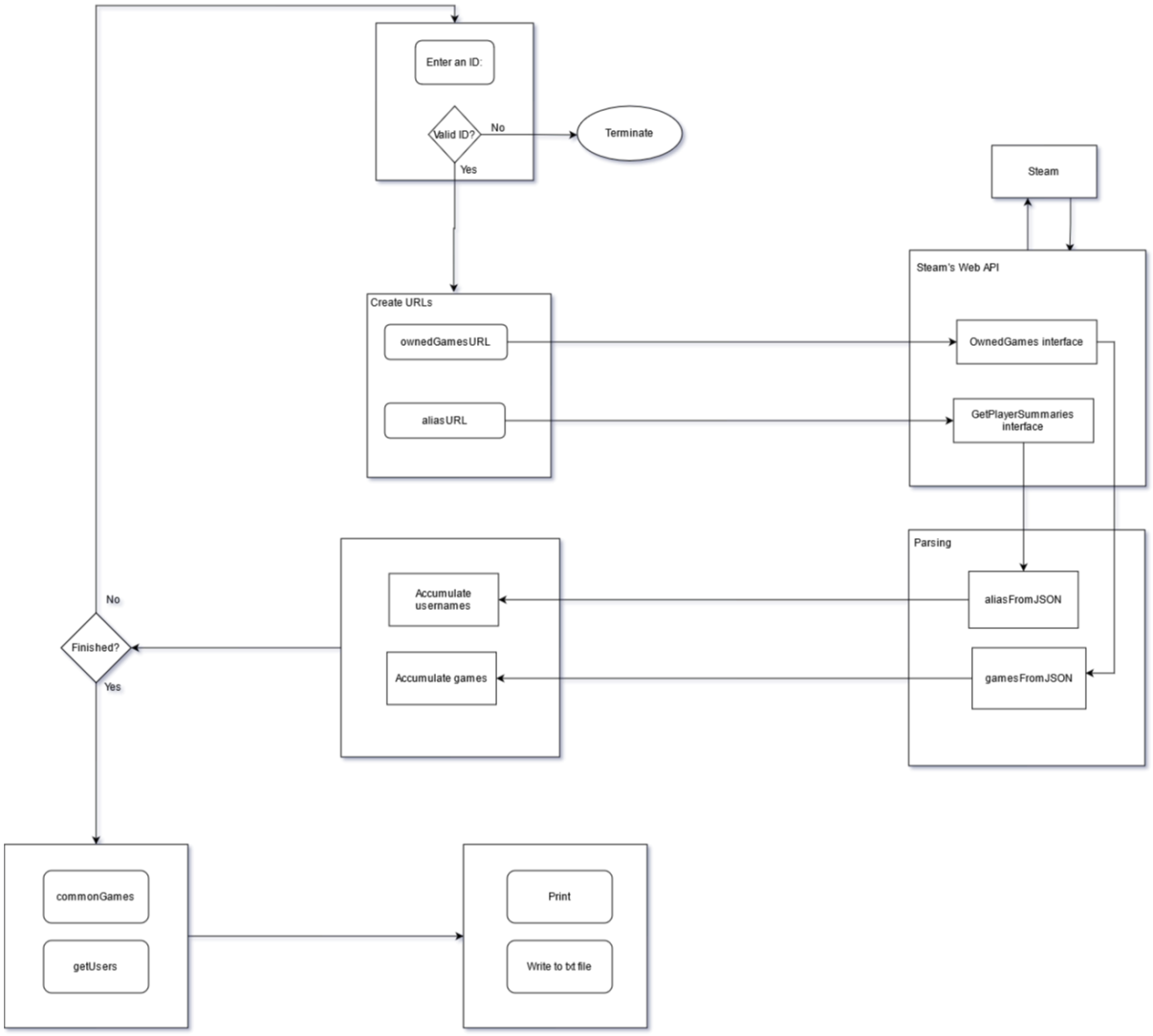
Once we have what we need from the JSON file, namely the games owned by the user in a list, we add that to another list that acts as an accumulator as we run the program recursively to gather as many Steam64 IDs as the user wants to compare. Once the user has gathered the Steam64 IDs and their games have been added to the accumulator, the user can type 'True' in order to signify that he has finished gathering Steam64 IDs. The program will then take this accumulator (to specify: this list filled with lists that have elements representing the games

owned by the Steam64 IDs) and send it to a recursive function that takes a list with a bunch of lists and basically looks for intersections between these lists and adds all these points of intersection to a new list that then gets returned. This new list is the list that contains all the common games of the Steam64 ID.

Main

The main function, main, is an IO function that starts out by printing "Welcome! Please enter a valid Steam64 to a PUBLIC Steam profile ... " then reads what you type. Once a Steam64 ID has been input. It will print "If you want to compare the following lists, type ‘True’ ". However, if the user types True with only one list, it will return that list as it checks for intersections with itself. What this means is that it checks for the games that it has in common with itself, namely returning a list of all the games owned. If the user however decides to add more than one Steam64 by typing anything other than ‘True’ it will call a new identical function to main but with one main difference. This new function takes the list of lists (which is created during the call of this new function in main by simply encapsulating the singular list of games provided by the first steam64ID into a list) as an argument, in order to remember the past steam64IDs.

Control flow



Discussion

Conclusion