I wrote an outline for this project early in the semester and built it out into reasonable pseudo code and code as I learned the necessary skills. Some of the code in this project is inspired by ways I accomplished tasks in weekly assignments, and other parts of it were things I learned while attempting to go deeper into/beyond the course material. This document provides details about the project including goals I set while writing it, choices I made, work I plan to do on it in the future, and acknowledgements.

Contents

Note About Perl Version: Throughout the semester I wrote most of my code on an old Mac that has Perl version 5.18.0. While working on this project I worked on a variety of machines:

1. Mac (old, small screen) — (Perl version 5.18.0)
2. Debian (laptop) — (Perl version 5.32.1)
3. Mac (new, bigger screen) — (Perl version 5.36.0)

The final version of the project was put together on the newer Mac and the Perl version used in the scripts for the project reflects that.

Modules Used:

1. Scalar::Util
2. Switch
3. Term::Menus
4. Term::ANSIColor (assumed this is what ANSI::Color in the project requirements meant)
5. Text::CSV
6. Other Modules I wrote (included in the project folder)

Project Goals:

1. **Use at least one module that was not required or suggested by the examples.**
   Status: Accomplished via `Scalar::Util`, `Text::CSV`.
   Motivation: I developed an allergy to packages, modules, and libraries after doing battle with Keras and Tensorflow during machine learning research projects; I wanted to get into the habit of seeing them as useful rather than frustrating.
   Learned: CPANM's ability to resolve dependencies is the eighth wonder of the world.

2. **Write subroutines when possible, and store those sub routines in separate files.**
   Status: Accomplished. Each menu and query type is stored in a subroutine.
   Motivation: Many of the programming courses on campus, especially C++ classes, cover the importance of splitting up code into separate files to make it more organized and easier to manage. I wanted to attempt accomplishing that in Perl.
   Learned: Accomplishing this goal required me to learn the basics of writing a Perl module.

3. **Experiment with different styles of Perl code and ways of doing things.**
   Status: Accomplished. A simple example is using scalars as file handles when opening/closing a file (as opposed to just `FH` as I did in assignments).
   Motivation: Becoming more fluent with Perl syntax and to begin testing the limits of TMTOWTDI.

4. **Write readable code without being obnoxious.**
   Status: Accomplished. The code is commented, but not overly commented, and variable names are readable without being as verbose as some of the ones I used in my assignments. Additionally, code is divided into subroutines but not to the point that it is overly compartmentalized.
   Motivation: Begin transitioning from writing for function (i.e. learning a concept) to caring about form too (i.e. more carefully balancing "I'm showing I understand how to do this thing and being clear about the mechanics" against concision/elegance).

5. **Use hashes as much as possible.**
   Status: Accomplished via hash and hash of hashes use.
   Motivation: The syntax of Perl hashes and their workings was one of my weak spots and I wanted to change that.
   Learned: I am more comfortable with Perl's hash syntax (though not perfectly comfortable) and I was able to experiment with how they can be used. For example, when considering the highest or lowest sales by genre, paltform, and publisher, and year. Hashes of hashes feel like using dark magic.

Project Files:

1. `jones_main.pl`: Main script file that should be run to launch the script.
2. `vgasales.csv`: CSV file that contains video game records (as provided).
3. `CSVCleanUp.pm`: "Cleaning" utility written to clean up the CSV file.

4. `ScriptMenus.pm`: Subroutines for the main menu and all submenus.
5. `TopSales.pm`: Subroutines for top sellers by genre, platform, publisher, and year.
6. `TopSalesYear.pm`: Subroutines for top sellers (for a given year) by genre, platform, publisher.
7. `HighestSales.pm`: Subroutines for highest sale by game, genre, platform, publisher, year.
8. `LowestSales.pm`: Subroutines for lowest sale by game, genre, platform, publisher, year.

Project Structure:

1. Upon launch the main script file "cleans" the csv file and writes the cleaned data to a new csv file.
2. The cleaned data csv file is then passed to the main menu subroutine to launch a menu with which the user can interact to make queries. The user is presented with four options:
   (a) Top Sales
   (b) Top Sales By Chosen Year
   (c) Highest Sales
   (d) Lowest Sales

   and each option calls a subroutine that displays a submenu for the option that was selected. The csv file is passed to the subroutine.
3. The user is asked to make a selection from the submenu for the query they would like to perform. The relevant subroutine is called for that query upon the user's choice. The csv file is passed to the subroutine so that the query can be performed on its data.
4. The user may be prompted for additional information needed to make the query (e.g. a year when selecting "top sales by chosen year").
5. The query is made and results are displayed to the screen.

Choices and Decisions: Specific logic choices are detailed in comments contained in each script/module file. The following descriptions are not the only exposition of the choices and decisions made.

1. **Looking through the CSV file to get familiar with the data.**
   Looking through the CSV file to get familiar with the data allowed me to notice that there was a blank field between the title and platform fields (i.e. the double comma), that some rows of the file were duplicates, and some of the years were entered as "N/A" instead of a year. This knowledge prompted me to write the `CSVCleanUp.pm` and informed how I chose to handle evaluating data in `TopSalesYear.pm` (detailed below).
2. **Loading data from the CSV file.**
   Loading data from the csv file is accomplished using a while loop, `Text::CSV`'s `getline()`, and pushing the rows into an array. This seemed to be the least painful and most familiar way to accomplish the task.

3. **Top sales for genre, platform, publisher, and year (TopSales.pm).**
   This is accomplished using a `foreach` loop that iterates through each element of the array into which the csv file's data was read. The element is "broken up" by its fields and each field is assigned to a relevant scalar (`$title`, `$platform`, etc) by dereferencing it with `@$`. Whatever is being queried (e.g. genre, platform, etc) is the primary key for the hash and within the hash are two additional hashes—one for the game title and one for sales figures. I am not sure if I wrote this in the best way, or even if I am describing it accurately (writing it was a result of "can I do this? [...] oh cool, it didn't break. [...] even cooler, the results look accurate!"), but this seemed to be the most straightforward way to accomplish my goal of using hashes when possible and keeping track of all the information I needed for the report printed to the user. It gets the job done!

   I check to see if the key does not exist or if the sales are greater than the sales currently recorded as highest sales, and if either of these are true the new data is recorded in the hash. Once the data is recorded in the hash the information is printed to the console using `printf` because it made it easy to print columns and use `Term::ANSIColor`'s functionality to bold and italicize text. No color is used in the results out of personal preference.

4. **Top sales in an entered year for genre, platform, and publisher (TopSalesYear.pm).**
   The code for identifying top sales in a user's entered year is similar to the description above (i.e. identifying top sales without regard to year), but we evaluate the year field of the element to see if it matches the user's entered year. To evaluate this I use the `looks_like_number()` function from `Scalar::Util` to deal with years listed as "N/A" in the csv file, and if the year field looks like a number and it is equal to the user's entered number we proceed as described above. If the year does not look like a number (i.e. is "N/A"), or the year does not match the user's entered year, the record is not evaluated further to see if it is the top sale for the given query.

5. **Highest and Lowest Sales (HighestSales.pm, LowestSales.pm)**
   I chose to break highest and lowest sales into separate options, but they are similar enough in how they work that they can be discussed together. In both modules I added an option, beyond what was asked for, to view highest/lowest sales by genre.

   Both use a hash with the query type (genre, platform, etc) as the key and the sales the value. Total sales are calculated by iterating through the sales fields in each element of the array and adding it to the sales value in the relevant key/value pair. To find the highest/lowest sales we create a scalar for the title and another for the sales (initialized to 0 if highest sales, or left uninitialized if lowest sales). Then we iterate through the hash using a `foreach` loop and evaluate if the value stored for sales is higher or lower than the currently recorded highest/lowest sales. If the evaluation is true the values for highest/lowest sales are replaced with the higher/lower sale data. The results are then printed to the screen with the name of the genre/platform/publisher/year and sales figure printed in bright green text.

   The highest/lowest selling game is handled differently than the other queries for highest/lowest sales. I chose to edit the code and calculate these differently because I realized that there were multiple games with the same lowest sales number (0.01) and it seemed inconsistent/incomplete to not list every game with those sales numbers. So the highest/lowest

sales for game subroutines break away from the other subroutines of this type to handle the possibility that there is more than one game to be reported. It accomplishes this by sorting the array that holds the data on the sales field, setting the highest/lowest sales according to the highest/lowest sales figure that exists in the array, finding all of the games that have this sales figure and storing them into a new array, and then sorting the new array alphabetically so that the printed report will be nicer to read.

Sample Output: For the sake of keeping this document more concise, a video demonstration of the script running on my machine has been included in this zip file. Its name is:

`jones_sp23_projectdemo.mov`.

Future Work: I plan to continue working on this script to make improvements. Improvements that I plan include:

1. **Proofreading the comments** carefully to make sure they accurately describe the query being done within the subroutine. Each subroutine was written using the first subroutine for the query as a model (i.e. `hs_platform` is a gently edited version of `hs_genre`), and it is very likely that there are some comments I overlooked editing.

2. **Standardizing the style** of the code seems like a fruitful thing to do. For example, in `CSVCleanUp.pm` when I open and close the file I include the error message with `$!`, but in other modules I do not do this—this reflects that I only truly appreciated the utility of including the error message while writing and testing the module (and that the utility was written later in the semester). I plan to go through the code to make sure I am consistent and using all possible useful features of Perl that are available.

3. **Rewriting other highest/lowest sales subroutines** to be like the highest/lowest sales for a game subroutines. I have not yet done enough analysis of the csv file to determine if editing the subroutines to account for the possibility that more than one genre, platform, etc, could be equally highest/ lowest selling, but it seems like it could be useful to do so.

Acknowledgments:

1. `https://www.perlmaven.com`. I consulted Perl Maven tutorials throughout the semester any time I had an idea and wondered whether something was possible.

2. `https://www.stackoverflow.com`. Stack Overflow, specifically

   (a) `https://stackoverflow.com/a/1712165`

   (b) `https://stackoverflow.com/a/46550384`,

   was extremely helpful to me when I was learning how to break my code up into different files (i.e. learning to write a module).

3. `https://perldoc.perl.org/perldsc#HASHES-OF-HASHES` was helpful while figuring out how to store information in `TopSales.pm` and `TopSalesYear.pm` subroutines.

4. `https://www.theunixschool.com/2013/05/perl-remove-duplicate-elements-from.html` helped me while I wrote `CSVCleanUp.pm`.

5. The documentation for all of the modules I used, as well as the official Perl documentation, was also helpful.