CS1112 Fall 2021 Project 5 due Monday Nov 14 at 11pm

You must work either on your own or with one partner. If you work with a partner you must first register as a group in CMS and then submit your work as a group. *Adhere to the Code of Academic Integrity.* For a group, "you" below refers to "your group." You may discuss background issues and general strategies with others, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not ok for you to see or hear another student's code and it is certainly not ok to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on you own, please seek help from the course staff.

Objectives

Completing this project will solidify your understanding of various types of arrays, including simple numeric array, simple `char` array, and cell array. You will also learn to process—read from and write to—data files. An additional objective of this project is to practice *problem decomposition*: you will break down a problem into several subtasks, based on your own design, and specify a subfunction for each subtask.

An adaptive study aid for testing your knowledge about popular songs

A "quiz system" is a common study aid for learning new vocabulary—think about a foreign language, or the vocabulary that you need to master in your Anatomy and Physiology class. Such quiz systems can range from paper flash cards to software tools that choose words to quiz you on according to your knowledge level. In this project you will implement such an adaptive quiz system for testing ones knowledge on popular musical artists and lyrics from their songs. Here is an example question:

```
Which of the following lyrics are by Bruno Mars?
1. Livin' la vida loca
2. Cause I'm Mr. Brightside
3. Who run the world? Girls.
4. And when you smile, the whole world stops and stares for a while
5. Hello from the other side
Type your choice here (0 to exit):
```

Our adaptive study system runs on a data set—a set of music artists and iconic lyrics from one of their songs—that is read from a plain text file. In addition to song lyrics, each artist in the data set has a rating, reflecting the artist's level of difficulty, and an estimated time-to-answer, which is the time it takes on average to answer the question (about that artist). A user is given a "user rating" to represent their knowledge level, and the system chooses an artist for the user to match some lyrics to based on the closeness of the user's rating and the artist's rating. Once the user answers the question, the system updates the statistics of both the user and the artist. In this way, the system "adapts" to the user's level. If you're an expert in popular songs, the system will tend to present harder questions; if you're not familiar with those exotic artists, the system will tend to choose easier questions.[1] Pretty cool!

Project overview

Our adaptive song study aid has these components:

- Main function `songStudySystem` is the "driver" of the system. It calls functions to read a data file, select questions and update statistics, and write a data file upon the user exiting the study session. *[Given]*
- Function `readSongData` reads the song data file and stores the data into a specified cell array. *[For you to implement]*

[1] Personalized learning is only one way of using such adaptive systems. Another use is for "learning" the "true" level of difficulty, or in general the true rating of something. For example, a vocabulary quiz system such as the one described here can be put online for many people to access. Then the artist ratings will evolve to a set of values that reflect the "true" relative difficulty of matching an artist to their songs.

- Function `singleQuestion` selects an artist and multiple candidate lyrics from the song data, presents the question and choices to the user, and accepts and evaluates the user response. This function also calls function `updateData` to update the relevant statistics. *[For you to implement]*
- Function `updateData` calculates the new user and artist ratings based on the user's answer and updates the relevant statistics. *[For you to implement with one helper function (subfunction) given]*
- Function `outputSongData` writes the song and session data to a data file. *[For you to implement]*

Download the file `p5_music.zip` from the course website. `p5_music.zip` contains all the functions named above (but most are incomplete), along with the song data file `songData.txt`. *Do not modify the specifications (function comments). You will complete/modify the function code as specified in the function comments.* You can run the given incomplete program by calling function `songStudySystem`, e.g.,

```
sessionStats = songStudySystem('songData.txt','songDataUpdated.txt')
```

The function will execute without any error messages, since most of the functions are empty at the moment you will see only dummy output. *As you implement the required functions one at a time, test each one by calling it the same way that* `songStudySystem` *does.*

The given driver function

Function `songStudySystem` is completely implemented for you; read it carefully. This function is the starting point of the program and drives the entire song testing session. You can read the flow of the session in the given code (and comments). Several constants and initializations in `songStudySystem` are further explained here:

- `baseKvalue` is a constant of the modified Elo rating system that is used for updating the user and artist ratings. We have provided the code for calculating the ratings in another function so you do not need to deal with this constant.
- `historyCount`=15 is the definition of "recently used" in our system, i.e., an artist is said to be "recently used" if it has been used in the previous 15 questions. Note that your code should still work correctly if historyCount changes to any positive integer value less than the number of artists in the dataset.
- `sessionStats` is a cell array of length 4 storing the statistics of the current song testing session. `sessionStats{4}` stores a vector of length 15. This vector will be used to store the indices of the previous 15 artists used in the current session, i.e., the indices of the "recently used" artists. Since no question has been asked at the beginning of a session, `sessionStats{4}` is initialized as a vector of 15 zeros. Consider this example session: the first question answered by the user was on the artist with index 101 and the second question answered was on the artist with index 36; then at that moment of the session `sessionStats{4}` would be a vector with the values 101, 36, followed by thirteen zeros.

Allowed functions, forbidden functions and actions

The data file is to be read into the system with the *provided* format—do *not* change the data file for reading it into the system. At the end of a song quiz session, you may update certain values when re-writing the data file but the format of the file must not change.

The only MATLAB types that we will use for handling text data are `char` and `cell`. *Do not use the type* `string`. Remember, use *single quotes* for type `char`; do not use double quotes.

You should use only the built-in functions that have been discussed within the course. Here are two functions we have not yet seen in the course that are useful for this project:

- `randi` - generate random *integer* values. Look it up in the MATLAB documentation (just type in the Command Window `doc randi`).
- `textscan` - its use in this project is explained below in §1. For future use be sure to check the MATLAB documentation because its usage is frequently modified from one release of MATLAB to another!

For file processing, you will need to use the functions `fopen`, `fclose`, `fgetl`, and perhaps `feof`, all of which you have seen in lecture.

You *must* <u>*not*</u> *use* the following functions: `randperm`, `randsample`, `find`, `strfind`, `contains`.

1 Getting the data

Implement function `readSongData` as specified in the function file. You can see the contents of `songData.txt` by double-clicking on the file name in MATLAB's *Current Folder* window. Read the function specifications to learn the format of the data file. The function specifications say that "within each line, data items are delimited by tabs." This means that two data items are separated by one tab in any line. Look at line 2 for example: it contains the characters '`200`', followed by a tab, followed by the characters '`1000`', followed by a tab, followed by the characters '`Mariah Carey`', followed by a tab, followed by the characters '`All I want for Christmas is you`'. The built-in function `textscan` is useful for parsing (separating) such tab-delimited text data. Assuming that you have used `fopen` to open the text data file and stored the file's identifer in a variable `fid`, here is an example for parsing any one line of the data file except the first, which has fewer data items:

```
L= fgetl(fid);  % read a line from the file with identifier fid; L is then a char vector
info= textscan(L, '%f %f %s %s', 'delimiter', '\t');
```

In the call to `textscan` above,

- the first argument is the `char` vector to parse, `L`;
- the second argument indicates the format of the individual data items in the `char` vector: `'%f %f %s %s'` indicates *number number string string*, just like the format sequences used in `fprintf` statements (note that a *string* here is just another name for *a sequence of characters*, not the type `string`);
- the final two arguments above say that the delimiter is a tab.

`textscan` returns a 1-d cell array of the data items, but a data item whose type is not `double` gets put inside a *nested* cell, i.e., a *nested* cell array is returned. For example, using the above `textscan` call to parse line 2 of the data file results in the cell array `info` like this:

```
{[200]}    {[1000]}    {1x1 cell}    {1x1 cell}
```

We do not want you to have to deal with nested cell arrays, so we have provided the function `cellstr2str` which removes the nesting of cells. The call `ca=cellstr2str(info)` would return the non-nested 1-d cell array of length 4 in `ca`:

```
{[200]}    {[1000]}    {'Mariah Carey'}    {'All I want for Christmas is you'}
```

Now you can access the data inside each cell of `ca`, for example,

```
t= ca{1};  % t stores the type double vector in cell 1 (the scalar 200 in this case)
r= ca{2};  % r stores the type double vector in cell 2 (the scalar 1000 in this case)
w= ca{3};  % w stores the char vector 'Mariah Carey'
d= ca{4};  % d stores the char vector 'All I want for Christmas is you'
```

Note that `ca{1}` is different from `ca(1)`; the former has the type `double` while the latter has the type `cell`. Generally, you need to access the *data inside* the cell wrapper and not the cell wrapper itself. *Use braces to access the data inside a cell.*

> For debugging, you may want to know the type (class) of a value. Use the function `class` to obtain the type name. For example, the call `class(ca{3})` from the example above would return `'char'`; the call `class(ca(3))` would return `'cell'`.

One of the return parameters of the function `readSongData` is the $n \times 4$ cell array `songData`, where n is the number of artists included in the data set. This cell array should be organized such that each row stores all the information for one particular artist, in the order of: artist, lryics from one song, time to answer, rating. For example, `songData{3,1}` stores the third artist, `songData{3,2}` stores lryics from one of the artist's songs, `songData{3,3}` stores the time taken to answer a question about the third artist, and `songData{3,4}` stores the difficulty rating of the third artist. Be careful: the order of the data in a row of cell array `songData` is different from the order of the data items in one line of the data file.

2 Asking the user one question and evaluating the result

Read the example session (screen output) shown on the course website; your program should produce screen output that has a similar format. Now read the specifications of the function `singleQuestion` carefully. This function covers every aspect of one question in a song testing session, from choosing the question to evaluating the user's response. *Decompose this problem by designing and implementing your own subfunctions (helper functions)!*

> *You are required to design and use at least two subfunctions* in this part of the project. You only *need* to design and use two subfunctions, but decomposing a problem into individual subtasks is an excellent design and program development strategy, so feel free to use more subfunctions to further modularize your solution. Make your subfunctions meaningful—don't just deal with this requirement superficially by creating a hardly necessary one-liner subfunction that does just a trivial calculation. Designing and specifying a useful (sub)function takes real effort, and it is exactly the *problem decomposition practice* and preparation that you need in order to deal with larger and more complex programs and problems in the future. Be sure to *fully specify* a subfunction by writing a concise comment that says what the subfunction does and defines the parameters.

Below we give more detailed information for implementing functions `singleQuestion` and `updateData`.

2.1 Picking the appropriate artist

What is an appropriate question to ask a user? Answer: a question based on an artist that has a rating similar to the rating of the user *and* the artist has not been used "recently," as defined by the constant `historyCount` in the main function `songStudySystem`. Here is the algorithm: for each artist that hasn't been used recently, calculate the difference between its rating and the user rating. The artist that has the smallest difference is the best choice. If multiple artists have the same minimum difference, choose one of them randomly such that each is equally likely to be picked.

Hint: We say "pick an artist," but what you really are picking is the *index* of the artist. The index is the row number of the artist in the cell array `songData`.

2.2 Generating the choices to present to the user

One of the choices must be the correct answer, and it should be placed in a random position among the `numChoices` choices that you present to the user such that the correct answer is equally likely to be choice 1, 2, ..., `numChoices`. The other `numChoices-1` choices must be randomly chosen from the remaining song data, each is equally likely to be chosen. The `numChoices` choices must be different from one another.

2.3 Timing the user

Start a timer (`tic`) immediately after presenting the choices and stop it (`toc`) immediately after the user inputs a value. See *Insight* page 52 for a discussion on `tic toc`. Store the time in tenths of a second since that is the unit used in `songData`. The time should be rounded to an integer value. (For example, if the user took 10.57 seconds to answer the question, you should store 106).

2.4 Evaluating the user's answer and updating statistics

You can assume that the user enters a numeric value as the answer. If the entered number is anything other than 1, 2, ..., `numChoices`, treat it as zero, meaning that the user wants to stop. If the user chooses to stop, then the question on the current artist has not been answered and it should not be included in any statistics—no updates. When updates are needed, call function `updateData` to perform the updates.

Read the specifications in function `updateData` carefully. Note that a subfunction `computeRating` is completely implemented; read the provided code to get a sense of how the modified Elo rating scheme works, but you do not need to know the details.

In `updateData`, calculate the new time-to-answer of the artist as a weighted average between the time taken in this session and the old time-to-answer: 70% of the time in this session plus 30% of the old time. Round the new time-to-answer to the nearest integer value.

Watch out! Remember that the vector in `sessionStats{4}` should have a fixed length of `historyCount`. If a session has more than `historyCount` questions, then before you can put the new data in `sessionStats{4}` you have to "make room" for it. Think carefully about how to maintain the correct history while keeping the vector the correct length.

3 Writing the output data file

Implement function `outputSongData` as specified.

4 Using your adaptive song quiz system

The example function call to `songStudySystem` shown on page 2 stores the post-session data in a file different from the original data. That is good for program development, when we do not want to change the original data. To use this system in the "adaptive" way, the output data file from one session becomes the input data file to the next session. Or simply use the same file name for both the input and output data file when you call `songStudySystem`. However, don't do this overwriting of the original data file until you have thoroughly tested your program and are certain that it is correct! Have fun learning which artists sing which lyrics. Challenge Dominic if you want to go up against the best.

Have fun with this! Feel free to change the artists and songs in `songData.txt`. You will not be submitting this file so you can add more lines to this file when you are quizzing your song knowledge. Take care to not change the format of each line in the file but change the first number in the file if you choose to add more artists to the file!

Submit your files `readSongData.m`, `singleQuestion.m`, `updateData.m`, and `outputSongData.m` on CMS.