CSCI-1200 Data Structures — Spring 2020
Homework 7 — Super Smash Bros. Frames

Overview

The creation of this assignment was inspired by Nintendo's popular fighting video game, *Super Smash Bros. Ultimate (2018)*. The game involves two or more fictional characters (which we will call *fighters*) battling each other with each character having unique *moves* they can use. As of when this assignment was written, there are 79 fighters available in the game. The purpose of this assignment is to be able to use maps and sets to build a small database that you can use to query or view "startup frame data" - how fast fighters can perform a given move - for each playable character in the game. No prior knowledge of the Super Smash Bros. series is needed to complete this assignment.

Startup Frame Data

As mentioned earlier, each fighter has various moves they can use to battle. Each move starts to happen at a different "frame". We will only be concerned with this type of frame data throughout this homework, also referred to as "startup frames" for each move.

Frame is the standard unit of time in a fighting game, where there are 60 frames in one second. If a fighter's move comes out at frame 60, it takes one second for the animation of that move to appear on your screen while playing the game; whereas if an option comes out at frame 3, the animation for that move would appear almost instantly. Below are a list of moves that each character can use in Super Smash Bros. Ultimate:

- jab
- forward-tilt
- up-tilt
- down-tilt
- forward-smash
- up-smash
- down-smash

For the sake of this assignment, assume that the above moves are the only ones a character may use, though in reality Super Smash Bros. Ultimate offers many more options.

Each character in the game will be able to use any of the above moves. However, the startup frames for each of these moves may differ by character. For example, both Pikachu and Kirby are playable characters in the game, so both of them are able to use up-tilt. However, the animation for Pikachu's up-tilt starts up at frame 7, while the animation for Kirby's up-tilt starts at frame 4.

We have provided two 'database' text files, `large_ult_frame_data.txt` and `mini_ult_frame_data.txt`, that contain frame data information for some or all playable characters in Super Smash Bros. Ultimate as of January 2020. You can assume that there are no errors in the formatting of these files.

As you work on this assignment, keep in mind that faster moves will have a smaller startup frame value and slower moves have larger startup frame values.

Queries & Output

To run your code for this assignment, use:

```
./smash.out [database_file] [input_file] [output_file]
```

The following commands may be present for any given [input_file]:

`-q [character_name] [move]/all`: A simple query - given the name of the character, output the startup frame data for one particular move or all moves of this character. If 'all' is specified, output should be in alphabetical order based on move name.

`-f [move] [limit]`: Output [limit] number of fighters that have the fastest [move], along with the corresponding frame data.

`-s [move] [limit]`: Output [limit] number of fighters that have the slowest [move], along with the corresponding frame data.

`-d [move] [startup frame]`: Output fighters that have a startup frame of [startup frame] for [move].

The output for these queries should be sent to [output_file].

You can assume that there are no errors in the formatting of [input_file], though it is possible that an illegal [character_name], or [move] is given; if this is the case, simply output

"`Invalid character name: [character_name]`"
or
"`Invalid move name: [move]`"

and move on to the next command. If both an invalid name *and* move are given, only print the error for the invalid name. You can also assume that every command starts with `-q`, `-f`, `-s`, or `-d` and that [limit] will be a non-negative integer.

For a query beginning with -f, -s, or -d, if two characters have the same startup frames for a particular move, break ties using alphabetical ordering based on character name. Please refer to the provided files for examples of this. For -f and -s if there are fewer than [limit] results, you should print as many results as you can.

Constraints

This assignment relies heavily on the usage of STL *map*s and *set*s - you are required to use at least one of each in your solution. In fact, they are the only containers you are allowed to use, along with *pair*s and custom classes that you write. You can use strings, ints, etc. but cannot use arrays (except for the global one in the provided `main.cpp`) and cannot use the *new* keyword.

The time complexity to check if a legal [character_name] or [move] is given should be $O(log\ (f))$ or $O(log\ (m))$ respectively, where f represents the total number of fighters provided and m represents the total number of legal moves. In addition to this, queries beginning with `-q` must look up fighters in $O(log\ (f))$ time as well. This constraint is placed to avoid iterating through an entire set of fighters or moves to check for illegal queries and data.

You are also required to create a *Fighter* class that will help organize information for each fighter.

Hints & Suggestions

The challenge of this assignment lies mainly in generating the *Fighter*s with the fastest or slowest moves in the game by only using maps and sets.

Then alphabetical queuering is taken care of

Think about which key-value pairs may be useful for retrieving information quickly. One suggestion would be to initialize a *map* where the key is a string representing a fighter's name, and where the value is a corresponding *Fighter* class object, similar to the student grades example in Lecture 15. After this map has been initialized, you can then use the stored information in it to create new maps or sets with different data types. We suggest that you write a function to populate this data structure by reading the contents of [database_file], and that you fill in this data structure before processing any queries.

Another suggestion is to overload the `operator<` for the *Fighter* class. You can then directly store *Fighter* class objects in an STL set based on the frame data value for a particular move. To accomplish this, you may need to create a member variable in your Fighter class that will keep track of the current method used for comparison (jab, forward-tilt, etc.).

Keep in mind that breaking ties occurs quite often in this assignment, both with respect to [character_name] and to [move].

Submission

Use good coding style when you design and implement your program. Don't forget to comment your code! Use the provided template `README.txt` file for notes you want the grader to read. You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.