COS226 – Term Project
Nathan Mathis, Crockett Lalor, Max Geffken

The Hearthstone Deck Builder is a utility application for fans of Blizzard Entertainment's video game Hearthstone: Heroes of Warcraft. The application serves two main purposes. The first of these is to act as an attribute-based search engine for all cards in the game. Upon starting the app, users are presented with a search bar, into which they can specify attributes to search for (the flags and search-able attributes are in the included README file). Hitting the search button populates a list with matching card names, and clicking through this list allows the user to view a picture some extra information about the selected card. The interface is meant to be fairly simple and easy to understand – as a result we mimicked in spirit the command language and argument specification of a typical command line program, such as git, nmap, or top, or any other a user might find and make use of in a typical GNU/Linux distribution. It also simplifies our work, and allows for much less interpretation/parsing logic than the alternative (which would be a Google-style search term). We also added the functionality for the user to add returned cards into a deck list. This gives the user of our application the ability to plan out decks and to tinker with the inclusion of different cards in their list.

The second purpose is to generate high quality decks for the user. The ranking of one card versus another is extremely subjective, and very rarely do two people agree on which cards are better than others in what situations – this is in fact where a lot of the skill in Hearthstone comes from and what makes hearthstone matches interesting/exciting. However, we did not have the time or determination to create a program to intelligently design decks in such a way as a human being would (to do so would be an impressive feat of engineering indeed), so we generate the decks based on a well-respected and heuristically determined so-called “tier list” of relative card rankings from http://heartharena.com/tierlist. The list is fairly accurate and there are many who rely on it when building Hearthstone Arena decks. Therefore, we deigned it “good enough”, and decided to build the deck generation around it (and the “Arena” game mode).

Arena is a game mode in which a player chooses a class, then 30 times, the player is given a choice of three cards, one of which he must add to his deck. Our plan was to simulate a draft, choosing cards according to their value. We found a tabulated representation of a tier list for each class online and loaded them into text files. Our plan was to use a hash table to map the names of cards to the tier values associated with them. We chose the hash table because, since the only use of the tier list part of the program was during drafting, accesses would be very flat across all our data, so it didn't make sense to use a splay tree, which would have absolutely been the right choice if users used the tier list directly, as there is a large class of cards players never look at. Also, a hash table has the most potential for later optimization, since tier lists change so little for relatively long periods of time, it would be easy to tweak the hash function and array size to yield true constant run time accesses.

In this final version of our project, only a few bugs remain:

Our most significant bug relates to the searchName() function, which, for some valid card names, does not return any results. Since draftDeck() chooses names from the tierList it builds from our text files, then queries the API to get the card information associated with that card name. Due to significant inefficiencies in our draftDeck() function related to rerolling all three cards being chosen from when only one of them fails to query the API properly, it can take from a minute and a half to two minutes to draft a deck. In addition, the draftDeck() function does not explicitly filter out cards of the wrong class, so some cards of the wrong class show up in the final deck list output. However, these errors, while important from a practical perspective, are unrelated to what we think is the important part of the drafting process, i.e., comparing cards by their class-specified tier value.

All things considered, our application is quite extensive and feature-rich. We even added some functionality that wasn't necessary but happened to make things slicker/more professional, like the ability to display every card's alternate “golden” art. In total, our program spans over 1100 lines of

code, not including the various text and xml files required by the application.

Notes:

The API we are using is omgvamp's hearthstone API, whose website is http://hearthstoneapi.com and whose mashape is https://market.mashape.com/omgvamp/hearthstone.

HTTP requests are made using the open source library Unirest, which can be found at http://unirest.io/java.

The build process, including resolution of dependencies, is managed by Maven, whose website is https://maven.apache.org

Our project is on github at https://github.com/NateM926/hearthstone_westwood

None of us had ever quite built a project like this, and merely getting our program to a state where it was actually consuming the API was actually quite difficult, and required quite a bit of tedious configuration and reading of Unirest/Maven documentation. Additionally, we had some trouble with version control. We decided on git early on, but didn't actually need or use it until later on in the process. It wasn't difficult to integrate, and git is simple and pleasant to use, but as one of our group members had never before used git at all, it was necessary to give him a sort-of crash course in git (which of course can be very confusing the first time through). Fortunately, we created a trello board to act as a kind of makeshift scrum board so that we could keep track of what needed to be done and where we were in our project. Because of the effort put into process and project management, everything thus far has gone quite smoothly.